

NAIST-IS-DD0661204

博士論文

ソフトウェア盗用を発見するための動的バースマーク

岡本 圭司

2007年 8月 23日

奈良先端科学技術大学院大学  
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に  
博士(工学) 授与の要件として提出した博士論文である。

岡本 圭司

審査委員：

松本 健一 教授 (主指導教員)

関 浩之 教授 (指導教員)

門田 暁人 准教授 (指導教員)

楢 勇一 准教授

# ソフトウェア盗用を発見するための動的バースマーク\*

岡本 圭司

## 内容梗概

今日、オープンソースソフトウェアの普及に伴い、公開されているソースコードを利用したソフトウェア開発が可能となった。その一方で、公開ソースコードの使用許諾条件（ライセンス）を遵守せず、密かに開発に用いるという新たな形態のソフトウェア盗用が問題となっている。このようなソフトウェア盗用では、盗用者はソースコードを公開しないため、盗用の事実を発見または立証することは非常に困難である。

この問題に対して、バースマークと呼ばれる、個々のソフトウェアに固有の特徴量を抽出し、盗用の発見に役立てようとする研究が行われている。しかし、ソフトウェアの静的な特徴量を用いる従来のバースマークは、Java プログラムには有効であるが、機械語プログラムに対しては利用することが難しい。Java プログラムでは、ソースコードの様々な特徴量が実行プログラムにも残存するため、実行プログラムから抽出したバースマークを用いて、ソースコード盗用の有無を検知できる。一方、一般的な機械語プログラムでは、コンパイラの種類が多く、コンパイラやコンパイルオプションの違いによって生成されるコードが全く異なるものとなる。しかも、コンパイルによってソースコードの特徴量の多くが失われるため、従来のバースマークを盗用の発見に利用することは難しい。

そこで、本論文では、ソフトウェアの実行時に API ( Application Program Interface ) の呼び出しを観測し、その情報を利用したバースマークを提案する。このようなソフトウェア実行時に得られる情報を用いたバースマークを「動的バー

---

\* 奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DD0661204, 2007年8月23日.

スマーク」と呼ぶ。一般に，OS が提供する API は，ファイル操作などの OS の機能を実行するための唯一のインターフェースであるため，あるソースコードをコンパイルして得られる機械語プログラムが呼び出す API の系列や API ごとの呼び出し回数は，用いたコンパイラの種類に関わらず，ほぼ同一のものとなると期待される。そのため，動的バースマークを機械語プログラムから抽出することで，ソースコード盗用の発見に有用である。

本論文では，提案バースマークの有効性を確認する 2 つの実験を行った。1 つめの実験では，提案手法はバースマークが備えるべき性質である保存性と弁別性を備えていることを確認した。2 つめの実験では，機械語プログラムが異なるコンパイラおよびコンパイルオプションによって作成された場合でも，提案バースマークへの影響は軽微であることを確認した。

また，提案バースマークの消去を試みる攻撃に対する耐性の評価を行った。提案手法は，従来から存在する難読化などの機械的変換ツールに対する耐性が期待できるが，人手による攻撃には弱点を持つことが分かった。そこで，編集距離を用いて API の挿入と削除による攻撃を無効にするバースマークの比較方法を提案し，評価実験によりその有効性を確認した。

## キーワード

ソフトウェア保護, ソフトウェアバースマーク, 電子透かし, ソフトウェア難読化

# Dynamic Software Birthmarks for Detecting Software Theft\*

Keiji Okamoto

## Abstract

Today's growth of open source software community has enabled rapid software development by reusing available open source code. Meanwhile, a new type of software theft - violating the license agreement by secretly reusing someone else's source code - has become a serious problem. Unfortunately, detection of such type of software theft is very difficult because thieves do not release their source code.

To detect a software theft, software birthmark has been proposed. A birthmark is a set of unique and native characteristics of a program. Conventional software birthmarks are effective for Java applications, but not effective for native machine language applications. Java applications hold their source code characteristics, and thus we can extract effective birthmarks easily. On the other hand, as for machine language programs, different compilers and optimization options produce quite different binary code, and this makes us difficult to extract stable birthmarks.

This dissertation proposes new software birthmarks using runtime API calls. The proposed birthmarks are called "dynamic software birthmarks" since they use runtime information of a program. It is expected that compilers do not change execution order and frequency distribution of API Calls. Therefore, it is useful to

---

\* Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0661204, August 23, 2007.

extract dynamic software birthmarks from a machine code program for detecting source code theft.

In the first experiment, we applied the proposed birthmarks to the same-purpose applications. As a result, it was shown that an application and its extended version have quite similar birthmarks, and that the applications that are independently implemented possess significantly different birthmarks. The second experiment showed that the proposed birthmarks achieve the strong resilience against the usage of different compilers and optimizations.

The experimental evaluation of the robustness against attacks have shown that our birthmarks survive existing automatic program translators such as obfuscators and optimizers, but messed up by an adversary's manual attacks. Thus, we proposed a new method to increase the robustness of our birthmarks, and we confirmed its effectiveness by an additional experiment.

**Keywords:**

Software Protection, Software Birthmark, Software Watermark, Software Obfuscation

## 関連発表論文

### 学術論文誌

1. 岡本 圭司, 玉田 春昭, 中村 匡秀, 門田 暁人, 松本 健一, “API呼び出しを用いた動的バースマーク,” 電子情報通信学会論文誌 D, Vol.J89-D, No.8, pp.1751–1763, August 2006.

### 国際会議

1. Haruaki Tamada, Keiji Okamoto, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto, “Dynamic Software Birthmarks to Detect the Theft of Windows Applications,” In Proc. International Symposium on Future Software Technology 2004 (ISFST 2004), October 2004. (Xi’an, China)

### 研究会・シンポジウム

1. 岡本 圭司, 玉田 春昭, 中村 匡秀, 門田 暁人, 松本 健一, “ソフトウェア実行時の API 呼び出し履歴に基づく動的バースマークの提案,” ソフトウェア工学の基礎 XI, 日本ソフトウェア科学会 FOSE2004, pp.85–88, November 2004.
2. 岡本 圭司, 玉田 春昭, 中村 匡秀, 門田 暁人, 松本 健一, “ソフトウェア実行時の API 呼び出し履歴に基づく動的バースマークの実験的評価,” 第 46 回プログラミング・シンポジウム報告集, pp.41–50, January 2005.

### NAIST テクニカルレポート

1. Haruaki Tamada, Keiji Okamoto, Masahide Nakamura, Akito Monden, Ken-ichi Matsumoto, “Design and Evaluation of Dynamic Software Birthmarks Based on API Calls”, Information Science Technical Report NAIST-

IS-TR2007011 ISSN 0919-9527, Graduate School of Information Science,  
Nara Institute of Science and Technology, May 2007.



# 目次

1. はじめに	1
2. 背景	6
3. 関連研究	12
3.1 ソフトウェア電子透かし（ウォーターマーク）	12
3.2 ソースコードの類似度による盗用の発見	12
3.3 著作者解析およびバースマークによる盗用の発見	13
4. 提案手法	16
4.1 提案手法のあらまし	16
4.2 準備	16
4.2.1 コピー関係	16
4.2.2 ソフトウェアバースマーク	18
4.2.3 バースマークの満たすべき性質	19
4.3 キーアイデア	20
4.4 提案バースマーク	21
4.4.1 実行系列バースマーク	21
4.4.2 実行頻度バースマーク	21
4.5 バースマークの類似度	22
4.6 実装の概要	23
4.7 実装の詳細	24
4.7.1 Step1: API 呼び出し観測ルーチンの挿入	24
4.7.2 Step2: API 関数ポインタテーブルの変更	24
4.7.3 Step3: API 呼び出しの記録	25
4.7.4 Step4: オリジナル API の呼び出し	26
4.7.5 Step5: バースマークの抽出	27
4.8 バースマーク抽出ツール – K2 Birthmark Tool Kit (K2BTK)	27

5. 保存性・弁別性の評価	29
5.1 実験1 バースマークの保存性, 弁別性の評価 . . . . .	29
5.2 実験2 コンパイラの最適化に対するバースマークの耐性 . . . . .	33
6. 攻撃耐性の評価	37
6.1 評価のあらまし . . . . .	37
6.2 既存のプログラム自動変換ツールに対する攻撃耐性 . . . . .	38
6.3 盗用者による攻撃に対する攻撃耐性 . . . . .	39
6.3.1 盗用者モデル . . . . .	39
6.3.2 攻撃の最小単位 . . . . .	40
6.3.3 盗用者の攻撃 . . . . .	40
6.3.4 攻撃のコストと攻撃可能回数 . . . . .	41
6.3.5 提案手法の攻撃耐性 . . . . .	43
7. バースマーク比較方法の改良	45
7.1 実行系列バースマーク比較方法の改良 . . . . .	45
7.2 編集距離 . . . . .	46
7.3 改良した実行系列バースマークの攻撃耐性 . . . . .	47
7.4 攻撃耐性のまとめ . . . . .	50
7.5 実験的評価 . . . . .	50
8. おわりに	52
参考文献	56
付録	63
A. 実験1の各ソフトウェアのAPI呼び出し回数	63
B. 7.5の実験的評価における実行系列バースマークの長さ	63

## 図目次

1	盗用の例 (a) . . . . .	2
2	盗用の例 (b) . . . . .	2
3	動的バースマーク . . . . .	18
4	保存性と弁別性 . . . . .	19
5	実行中のソフトウェアに DLL を挿入 . . . . .	25
6	API 呼び出しの観測 . . . . .	26
7	導出された API 系列の例 . . . . .	28
8	実行頻度バースマークのグラフ . . . . .	32

## 表目次

1	実験 1 で使用したソフトウェア . . . . .	30
2	実行ファイルの差分サイズによる類似性評価 . . . . .	30
3	実験 1 結果: 実行系列バースマーク (EXESEQ) による類似度評価	31
4	実験 1 結果: 実行頻度バースマーク (EXEFREQ) による類似度評価	31
5	実験 2 で使用したコンパイラ一覧 . . . . .	34
6	実行ファイルのサイズ . . . . .	34
7	実験 2 結果: 実行系列バースマーク (EXESEQ) による類似度評価	35
8	実験 2 結果: 実行頻度バースマーク (EXEFREQ) による類似度評価	36
9	攻撃の分類 . . . . .	41
10	攻撃コストの内容 . . . . .	43
11	攻撃コスト . . . . .	43
12	提案手法の攻撃耐性 . . . . .	44
13	編集距離算出アルゴリズム (1) . . . . .	46
14	編集距離算出アルゴリズム (2) . . . . .	46
15	編集距離 セルの値の算出式 . . . . .	47
16	編集距離算出アルゴリズム (3) . . . . .	47
17	API 関数削除攻撃の模式図 . . . . .	48

18	API 関数挿入攻撃の模式図 . . . . .	48
19	API 関数置換攻撃の模式図 . . . . .	49
20	API 関数順序入替攻撃の模式図 . . . . .	49
21	改良実行系列バースマーク 実験的評価の結果 . . . . .	51
22	各ソフトウェアの API 呼び出し回数 . . . . .	63
23	実験的評価における実行系列バースマークの長さ . . . . .	63

## 1. はじめに

今日、コンピュータとソフトウェアによる情報システムが担う社会的役割はますます高まっている。その片翼を担うソフトウェアは、従来ではプロプライエタリなソフトウェアが中心であったが、今日では GPL[1] に代表されるようなオープンソースライセンスによるソフトウェアが重要な位置を占めている。その中で、ソフトウェア盗用の問題は、従来から存在するソースコードの流出と改ざんだけではなく、オープンソースソフトウェアのライセンスに違反するソースコード利用（盗用）が問題となっている。

開発時のソフトウェアにおける権利の取り扱いは、使用許諾書（ライセンス）や契約によって行われている。GPL に代表される、オープンソースのライセンスのほとんどでは、受け取ったソースコードを、そのままの状態であれば再配布は許可される。また GPL では、受け取ったソースコードを改変しても、同じ GPL で公開するのであればライセンス違反に問われることはない。

しかし、開発者の制約はライセンスで定められているだけで、ソフトウェアやソースコードが物理的に保護されているわけではなく、簡単に盗用することができる。また、ライセンスに違反し、密かに別のソフトウェアに利用（盗用）したとしても、ライセンス違反を他者が知ることは必ずしも容易ではない。このため、ソフトウェアをライセンス違反して使用することに対する技術的・心理的な障壁は高くないことが現状である。現実にライセンス違反の事例は数多く報告されている。例えば、ソースコードが公開されていたミニツツマスコットが盗用され PocketMascot として公開されていた事例 [2] や、Mac OS X エミュレータの CherryOS がオープンソースの PearPC のソースコードをライセンス違反して盗用していた例 [3]、ELECOM のブロードバンドルータに GPL のソフトウェアが組み込まれていたがソースコードが公開されていなかった例 [4] などがある。

ソフトウェア盗用の例を図 1 および図 2 に示す。プロプライエタリなソフトウェアの場合、(1a) 悪意ある開発者が会社 A からソースコードを入手し、(2a) そのソースコードを改変し別のソフトウェアとして見せかけて、(3a) 会社 B のソフトウェアとしてバイナリを販売する（図 1）。オープンソースソフトウェアの場合では、(1b) 悪意ある開発者がインターネットなどを通じてソースコードを入手し、

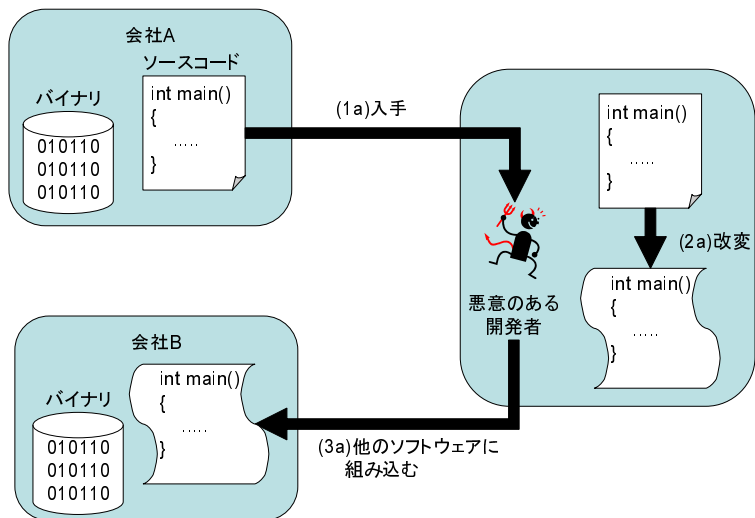


図 1 盗用の例 (a)

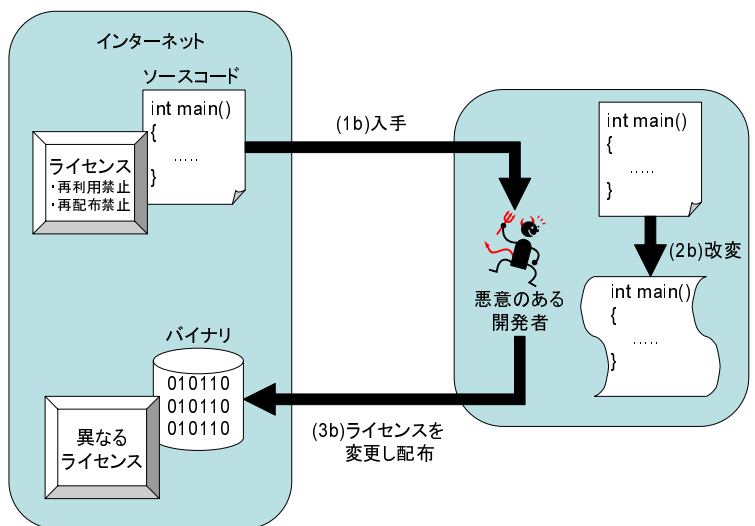


図 2 盗用の例 (b)

そのソースコードを改変し，別のソフトウェアと見せかけ，(3b) 新たなライセンスでバイナリを配布する（図2）．

上記2つの例で，(1a) および (1b) に存在するソースコードと，(3a) および (3b) で配布されるソフトウェアは一見何の関係もないように見えてしまう．ソースコードが公開されている場合には，コードクローンの検出技術 [5] などで検出することができるが，バイナリプログラムのみが配布されている場合には，ライセンス違反の有無を確認することは難しい．以上のことから，ソフトウェアのバイナリのみを用いて，盗用の事実を発見・立証する技術が求められている．盗用を発見する技術が確立すれば，安易な盗用に対する強力な抑止力になることが期待できるからである．

これらの問題に対して，現在ウォーターマーク（電子透かし）と呼ばれる技術が提案されている．ウォーターマークは，ソフトウェアを書き換えて内部に著作権情報を埋め込み，必要なときに取り出すことで，盗用を発見および立証する技術である．ウォーターマークは，バイナリプログラム全体の盗用や，Java プログラムにおけるクラスファイル単位での盗用，外部ライブラリの盗用などに対しては有効な技術である．しかし，ウォーターマークはあらかじめ埋め込んでおく必要があるため，盗用が発覚した後では利用することはできない．また，オープンソースソフトウェアの場合では，ソースコードに人知れず透かし情報を埋め込んでおくことは難しく，発見された場合には取り外されるおそれがある．

一方，バースマークと呼ばれる技術が提案されている．バースマークは，ソフトウェアの実行に不可欠な部分からそのソフトウェアの特徴を抽出し，それを手がかりにソフトウェアを識別して盗用を発見する技術である．ウォーターマークと比較すると，事前に情報を埋め込んでおく必要がない・比較対象ソフトウェアのソースコードがなくても利用できる・複数のバースマークを同時に使用することができる，などのメリットがある．バースマークは，Java プログラムを対象に広く研究され，有効性が確認されている [6]．従来のバースマークは，静的バースマークと呼ばれ，ソフトウェア（実行ファイル）の静的解析によって得られる．Java プログラムでは，計算機環境やコンパイラの違いに関わらず，ソースコードに含まれる特徴量がバイナリプログラムにも残存するため，ソースコードの盗用

を発見するためのバースマークを容易に構成することができる。

しかし、機械語を使用する一般的なコンピュータアーキテクチャ上のバイナリプログラムでは、ソースコードに含まれる特徴量の多くが失われている。同じソースコードをコンパイルしてバイナリを作成する場合でも、異なるコンパイラやコンパイルオプションでは、それぞれ異なるバイナリが作成される。実用的な逆コンパイラが存在しないことから、ソースコードとバイナリの非対応性は明らかである。このため、Java プログラムのようにソースコードの盗用をバースマークで発見することは困難であり、これまでに機械語プログラムを対象としたバースマークは提案されていない。

そこで本論文では、機械語プログラムを対象とし、ソフトウェア実行時に API (Application Program Interface) 呼び出しを観測して得られる情報を利用したバースマークを提案する。キーとなるアイデアは以下の通りである。OS 上で実行されるソフトウェアは、例えばファイルの入出力などは OS の提供する API を用いてのみ可能な操作である。つまり、API 呼び出しを別の命令に置き換えるなどして簡単に消去することはできないため、バースマークとして用いる情報として有用と考えられる。しかし、API を呼び出すための機械語は多くのパターンが考えられるため、API 呼び出しを実行ファイルから静的に読み取ることは難しい。そこで、実行時に API 呼び出しを観測しその情報をバースマークとする。提案バースマークは、ソフトウェアの静的な情報を用いた静的バースマークに対して、実行時の動的な情報を用いることから、動的バースマークと呼ばれる。

提案手法は、対象ソフトウェアのソースコードが入手できなくても利用可能で、バイナリのみを用いてソースコードの盗用を発見することができる。また、Windows 環境をはじめとした機械語を使用するコンピュータアーキテクチャ上で使用できる。本論文では、提案バースマークを Windows 環境で実装する方法を示す。また、提案バースマークの有効性を、実験を通じて評価する。

提案手法は、初めて具体的に実装された動的バースマークである。また、オペレーティングシステムの中でも最大のシェアを持つ Windows 環境を対象とした、初めてのバースマークである。

本論文の構成は以下の通りである。第 2 章では研究の背景について、第 3 章で



は関連研究について述べる．第4章では，提案手法のキーアイデアと2種類の提案バースマーク，そして提案手法の実装方法について述べる．そして，第5章では，提案手法の保存性と弁別性を評価する実験とその結果について述べる．第6章では，提案手法の攻撃耐性について議論する．第7章では，第6章の議論を受け，より攻撃に強いバースマークの設計を試み，実験的評価を行う．最後に第8章を本論文のまとめとする．

## 2. 背景

今日，ソフトウェアの盗用が世界中で深刻な問題となっている．典型的な例として，以下の3つが挙げられる．

- (1) アプリケーション全体をコピーし，販売する（不正コピー）
- (2) アプリケーションの一部（モジュールやコードの一部）を盗み，許可なく他のアプリケーションの一部として使う
- (3) オープンソースソフトウェアのソースコードを不正に使用する（ライセンス違反）

(1) は，典型的には海賊版や不正コピーと呼ばれる被害で，ソフトウェアのバイナリがそのままもしくはライセンス認証部分を無効にするなどの改変が行われた上で，不正に販売もしくは配布される被害である．(2) は，典型的な例としては，転職したソフトウェア開発者が，前職の職場からソースコードをコピーし，新しい職場で利用するものである．この場合，ソースコードの入手そのものが違法に行われている．(3) は，今日のオープンソースライセンスによるソフトウェアの普及により，非常に多く報告されるようになった盗用である．この場合，ソースコードの入手は合法に行われたが，違法に利用されているものである．上記の例のうち，本論文で解決したい盗用は(2)と(3)である．

ソフトウェアの盗用に関して，これまで多くの事例が報告されている [7]．ソフトウェアの盗用として著名な事件を以下に挙げる．なお，事件の調査にあたっては，奈良先端科学技術大学院大学特任助教の玉田春昭氏に全面的に協力いただいた．

ミニッツマスコットのソースコード盗用事件 [2] 上野智弘氏によって作成された，PDA 上で動作するマスコットソフトウェアであるミニッツマスコットは，ソースコードを含めて広く公開されていた．その後，Janusz Gerszberg という人物が，Pocket Mascot というソフトウェアを公開した．Janusz Gerszberg は，以前から上野氏に対し「共同で，ミニッツマスコットを有償のソフト

ウェアとして公開しよう」と提案しており，上野氏はそれを拒否していた．上野氏が，PocketMascot はミニッツマスコットの盗用であると抗議したところ，PocketMascot の公開は停止された．

JM5 のソースコード盗用事件 [8] JM5 は河邊径太氏らが開発した，Psion と呼ばれる PDA 上で動作する電子メールソフトウェアである．あるとき，Psion PDA を輸入販売している有限会社パックスシステムズから，河邊氏らに「JM5 を PDA にバンドルし販売したい」と連絡があった．河邊氏らは，いくつかのやりとりの後それを拒否した．その後，パックスシステムズは，J Mail というソフトウェアを販売開始した．J Mail は JM5 と同様に Pison 環境で動作する電子メールソフトウェアだった．河邊氏らは，JM5 と J Mail の類似性について検証したところ，J Mail は JM5 のメール送信部を流用し，いくつかの機能を付け足して作成されたと確信した．この点についてパックスシステムズに抗議したところ，パックスシステムズは「開発者が退職し詳細は分からない．今後 J Mail の販売は行わない．さらに，Psion PDA の販売および Psion 環境のソフトウェアの開発販売から撤退する」との回答があった．

ネットゲーセンのライセンス違反によるソースコード盗用 [9] 株式会社メディアカイトは，ネットゲーセンと呼ばれる，インターネットを介して過去のアーケードゲームを配信し，自宅でプレイするシステムを提供すると発表していた．そして，ネットゲーセンで使用するとされていた，OAGC システムと呼ばれるソフトウェアは海外の別会社が開発していた．OAGC システムそのものはすでに入手可能な状態だったため，熱心なゲームユーザが OAGC システムを入手したところ，MAME と呼ばれるソフトウェアと類似性が高いことが判明した．そして，インターネット上の複数のサイトでは，検証により「OAGC システムは MAME のソースコードを盗用している」との指摘がなされた．その後，メディアカイトは，ネットゲーセンで使われるシステムの一部に著作権侵害の恐れがあるとして，ネットゲーセンの無期延期を発表した．

家電店がオープンソースソフトウェアを“ 自社開発 ”と偽って販売した事件 [10][11]

ある家電店が、NEC 製ハードディスクレコーダー AX-10 上で動作するソフトウェア wizr on NEC AX home AV server (以下、wizr on AX) を AX-10 にバンドルし販売していた。しかし、wizr on AX はオープンソースソフトウェアとして公開されているソフトウェアであり、そのライセンスでは二次配布および商用利用等を禁止していた。家電店は、wizr on AX を自社開発と偽った上で、サポート用の問い合わせ窓口として作者のメールアドレスを記載していた。サポートを求めるメールが殺到し、無断で販売されている事実を知った作者が、家電店に問い合わせたところ、「そんな事はない」「アルバイトが勝手に企画したことなのでわからない」「これで有名になったんだから良かったと思ったほうがいい」「ユーザーサポートの費用払ってやってもいい」「その代わりソフトの権利はうちの会社でもらう」「所詮タダで配っているソフトだから誰の著作権も何もない」などと回答した。

フリーソフトウェアを無断盗用し自作ソフトウェアであるとして再公開した事件 [12]

ある作者が公開していたフリーソフトウェア RegistryTunerXP は、同種のソフトウェア NTREGOPT の盗用ではないかという疑惑が持たれていた。RegistryTunerXP はオンラインソフトウェア配布サイト Vector で配布が行われていたため、有志がこの件について Vector に問い合わせたところ、Vector は盗用の疑いが強いことを確認し、RegistryTunerXP の配布を停止し、作者にその旨を通告した。その後、この作者は多数のソフトウェアを盗用し、自作であるとして公開していることが判明した。この作者は、ソフトウェア中に含まれる文字列を書き換えることで、異なるソフトとして配布していた。

さらに、GNU GPL (GNU General Public License) [1] と呼ばれるソフトウェアライセンス違反の事例が多数ある。GPL の下で提供されるソフトウェアは、ソースコードが提供されており、自由に利用することが可能である。ただし、GPL により公開されているソースコードを利用したソフトウェアは、同じく GPL ライセンスにより公開しなければならない。つまり、GPL によるソースコードを利用した場合、そのソフトウェアはソースコードを公開しなければならない。また、

GPL には著作権表示を削除してはならないという規定も存在する。GPL 違反の例としては以下のようなものがある。

プロジー株式会社の DivX コンバータ [13] プロジー株式会社（現在は株式会社ライブドア）が販売していた DivX コンバータは、インターネット上の複数のサイトで GPL 違反を指摘された。DivX コンバータは、DeCSS、DVD2avi、bbMPEG、Lame の GPL で公開されている各ソフトウェアのソースコードを利用して開発されていた。プロジーは GPL 違反を認め、DivX コンバータのソースコードを公開した。

PoneView の事例 [14] ある作者の公開するソフトウェア PoneView は足永拓郎氏の開発したソフトウェアである GImageView のソースコードを利用していたが、オリジナルの著作権表示を削除していることが指摘された。そこで、PoneView は著作権表示を元に戻した上で再公開された。

Sigma Designs 社の REAL MAGIC MPEG-4[15] Sigma Designs は REAL MAGIC MPEG-4(RMP4) と呼ばれる動画圧縮のためのソフトウェアを公開したが、GPL で配布されているソフトウェア XVID のソースコードを流用しているのではないかと指摘された。Sigma Designs は流用を認め、RMP4 のソースコードを公開した。

エプソンコーワの Image Scan! for Linux[16][17] エプソンコーワ（現在はエプソンアヴァシス）は、Linux 上でセイコーエプソン社製スキャナのドライバである、Image Scan! for Linux を無償配布していた。しかし、同ソフトウェアでは、GPL で公開されている gettext のソースコードをパッケージに取り込みながら、ソースコードを非公開にしていた。エプソンコーワは、使用する gettext パッケージを LGPL 準拠の 0.10.40 に差し替える、LGPL 2.1 のセクション 6 に明記されている目的に限りリバースエンジニアリングを許可するよう使用許諾を変更するなどの変更を行い、再公開した。この事例では、エプソンコーワが日頃からオープンソースコミュニティに協力的であったこと、ライセンス違反指摘後の対応も迅速であったことなどから、オープンソースコミュニティからは賛辞の声が上がった。

ELECOMのブロードバンドルータ [4] ELECOM社製のLD-BBR/Bをはじめとするブロードバンドルータ製品は、LinuxなどのGPLソフトウェアが含まれていたが、ソースコードが公開されていなかった。有志がこれを指摘したところ、当初「ソースコード公開の必要はない」などと対応していたが、オンラインコミュニティでの議論が盛り上がるにつれ方針を変更し、ソースコードを公開するに至った。

MacOS X エミュレータのCherryOS[3][18] MXS社は、MacOS XをWindows上で動作させるためのエミュレータソフトウェアCherryOSを公開した。しかし公開後、同種のオープンソースソフトウェアPearPCのコードを流用しているのではないかと疑念が持ち上がった。有志がCherryOSとPearPCに含まれる関数名や文字列などを比較検証したところ、PearPCから相当量のコードを流用していると指摘された。また、PearPCの作者は、PearPCに含まれる特定の意味を持たない変数名を、CherryOSが持っているとし、CherryOSの盗用を指摘した。MXS社は盗用に反論し、CherryOSの新バージョンを公開するなどしたが、新バージョンでも相変わらず盗用の疑いが指摘された。MXSは、盗用を認めないままCherryOSのオープンソース化を宣言したのち、最終的には開発中止を発表し、ソースコードは公開されなかった。

また、ソフトウェアの盗用を巡って裁判に至った例も存在する。

1. Symantec が、プログラムコードを盗用したとして McAfee を提訴した事件 [19]
2. シビルソフト開発が、土木設計ソフトのコピー販売を行ったとして、損害賠償を求めてビーアイテイを提訴した事件 [20]
3. SCO Group が、ソースコードのライセンス違反を理由に IBM を提訴した事件 [21]

ソフトウェアの盗用はそのソフトウェアを販売する会社に深刻な被害を与えることになる。しかし、ソフトウェアを盗用から守ることは容易なことではない。

その理由は2つ挙げられる。第1に、ソフトウェアの一部が盗用された場合に、それを発見するのは容易ではない。ソフトウェアの全体が盗用された場合にそれを発見するのは比較的容易であるが、一部が盗用された場合にはプログラムの挙動やGUIデザインなどが大幅に変わるため、発見するのが難しい。第2に、もし盗用が疑われるソフトウェアを発見した場合でも、盗用の事実を法廷で立証することは容易ではない。盗用を立証するためには、相互のソフトウェアの類似性を明らかにする必要があるが、ソフトウェアの類似性に関する技術は確立されていない。特に、ソフトウェアが盗用され、そのソースコードが入手できない場合の立証は非常に難しい。同じソースコードを用いても、異なるコンパイラやコンパイルオプションで全く異なるバイナリが作成されるためである。

## 3. 関連研究

### 3.1 ソフトウェア電子透かし（ウォーターマーク）

ソフトウェアに対する電子透かし（ウォーターマーク）は盗用されたソフトウェアの著作権を証明するための技術としてよく知られており，盗用の疑いのあるソフトウェアを発見するという目的にも用いられる．この技術はソフトウェアの一部に著作権情報やユーザ ID 等の情報を密かに埋め込んでおき，必要なときにその情報を取り出し，盗用の立証に役立つものである．情報を埋め込む方法は，静的透かしと動的透かしに大別できる．静的透かしは，プログラムの一部を書き換えることで情報を埋め込み [22, 23]，動的透かしはある入力  $I$  が与えられたとき，プログラムの実行中に透かしが出力されるようにプログラムを書き換える [24, 25, 26]．

電子透かしはあらかじめ情報を埋め込んでおかなければならないため，過去にリリースした（透かしの入っていない）プログラムに対しては効力を持たない．また，複数のモジュールから構成されるプログラムでは，全モジュールに透かしの埋め込むことが望ましいが，大規模なプログラムでは，全モジュールに透かしの埋め込むことは困難である．また，全てのモジュールが透かしの埋め込むために十分なサイズを持っているとは限らない．さらに，透かしとして埋め込む情報は，プログラムの実行という視点から見ると無駄な情報であるため，最適化や難読化などの等価変換を施すことで消滅したり，人手によって改ざんされる可能性がある．オープンソースソフトウェアの場合には，ソースコードに人知れず透かし情報を埋め込んでおくことは難しく，発見された場合には取り外されるおそれがある．

### 3.2 ソースコードの類似度による盗用の発見

ソースコードの類似度を測ることで盗用を発見する技術が提案されている．これらは，プログラミング教育の際に用いられることが多い [27, 28, 29, 30]．プログラム教育での盗用とは，プログラムの提出を課された課題において，他者の作



成したプログラムをそのままコピーして提出する，もしくは変数名や関数名の変更など，小規模な変更を加えた上で提出するものである．この盗用を発見する手法としては，ソフトウェアメトリクス [27, 29, 30] や Kolmogorov 複雑度 [28]などを元に類似度を計算する手法などがある．これらの手法は，ソースコードが入手できることを前提としているが，ソフトウェアの盗用の場合ではソースコードが手に入らないことが多く，利用することが難しい．また，難読化や最適化などツールによる攻撃も考慮されていない．

上記の手法のほかには，コードクローンの計測技術を用いて盗用を発見する方法が提案されている [31, 5]．コードクローンとは，プログラムのソースコード中に存在するコード片で，他の部分からコピー&ペーストで作られた部分や，よく似た形や構造の部分のことである．もし，全く別のソフトウェアから大規模なクローンが発見された場合，どちらかのソフトウェアがコピーにより作られた疑いが非常に強い．神谷らは，コードクローンを発見するためのツール CCFinder を使って，FreeBSD の `sys/net/zlib.c` と Linux の `drivers/net/zlib.c` の2つのソースコードがほぼ同じであることを発見した [5]．コードクローンは，コピーの疑いのあるソフトウェアを見つけるために有用であるが，ソフトウェア盗用に対しては，やはりソースコードの入手が前提であること，難読化や最適化などが考慮されていないことなどが問題となる．ソースコードが入手できない場合にコードクローン発見技術を適用するためには，バイナリのアセンブリ表現を利用することになるが，アセンブリ表現はコンパイラの違いやコンパイルオプションの違い，ソースコードの小さな変更に対しても敏感に変化するため，盗用の発見のために利用するのは困難となる．

### 3.3 著作者解析およびバースマークによる盗用の発見

オブジェクトコードからソースコードの著作者を特定するための技術が提案されている [32]．Krsul と Spafford はプログラムのスタイルメトリクスやレイアウトメトリクスを用いてこれを実現する方法を提案した [33]．また，Spafford と Weeber はウィルスやトロイの木馬の残骸からデータ構造やアルゴリズム，システムコールの好みを用いて作者を特定する方法を提案した [34]．

また、バースマークと呼ばれる技術が提案されている。バースマークは日本語に直訳すると「あざ」という意味であり、ソフトウェア指紋と呼ばれることもある。バースマークは、ソフトウェアの実行に不可欠な部分の集合と定義される。バースマークは、ソフトウェアに対する電子透かしと比較して、あらかじめ情報を埋め込む必要がないことが特徴である。つまり、バースマークは盗用が発覚してからでも用いることができる。また、複数の手法を同時に用いることができ、個々のバースマークの欠点を補い合ったり、より強固な立証とすることができる。

玉田らは Java クラスファイルを対象としたバースマークを提案した [35, 36, 37, 6, 38]。玉田らの手法はプログラム中の特徴的な箇所である初期値代入部分、継承関係、メソッド呼び出し部分、依存クラスを静的解析により抽出し、識別に用いる。しかし、玉田らのバースマークに対する攻撃法が福島らにより指摘されている [39]。福島らは、Java プログラム中の各メソッドからオートマトンを抽出することで、バースマークとする手法 [39] や、Java プログラムのメソッドから状態遷移図を抽出し、それに対してパラメータを与えたときの実行系列をバースマークとする手法を提案している [40]。Myles らは、Java プログラムの命令列から k-gram を抽出しバースマークとする手法を提案している [41]。これらの手法はソフトウェア（実行ファイル）の静的な情報を用いるため、静的バースマークと呼ばれる。Java プログラムでは、計算機環境やコンパイラの違いに関わらず、ソースコードに含まれる特徴量の多くがバイナリにも同様に含まれるため、これらの情報を用いることで有効な静的バースマークを構成することができる。しかし、一般的なコンピュータアーキテクチャ上の機械語プログラムでは、ソースコードの特徴量の多くが失われており、また、同一のソースコードを用いても、コンパイラやコンパイルオプションで生成されるバイナリプログラムが全く異なるものになるため、これら静的バースマークの手法をそのまま用いることは難しい。また、一般的にプログラムの静的解析により得られたバースマークは、バースマークとして抽出している特徴量を特定しやすく、改変による攻撃に弱い傾向にある。

これに対して、我々の提案と同時期に、Myles らは Java プログラム実行時の経路情報を用いた Whole Program Path (WPP) と呼ばれるバースマークを提案している [42]。静的バースマークに対して、ソフトウェア実行時の情報を用いた

バースマークを動的バースマークと呼ぶ。WPP は、ソフトウェアの実行経路情報を用いることから、ループの平坦化など最適化技術に弱いと考えられる。さらに、機械語プログラムでは、その実行経路情報を収集することは非常に困難である。従って、WPP は機械語プログラムに対しては利用できないと考えられる。

## 4. 提案手法

### 4.1 提案手法のあらまし

3章で述べたように，既存のウォーターマークやバースマークは，ソースコードの盗用に対して利用できない・機械語プログラムに対して利用できない・改変に弱いなどの問題が存在する．そこで本論文では，これらを解決する新たな2種類の動的バースマークを提案する．一般的に，中規模以上のソフトウェアはOSが提供するAPI (Application Program Interface) を用いて作成される．これは，OSが提供する機能を積極的に利用することで，容易にソフトウェアを作成できるからである．また，システム保護のため，いくつかの機能(ファイルの入出力など)は，特定のAPIを用いてしかアクセスすることができない．このようなAPIは，等価な別の命令に置き換えることはできない．また，APIの呼び出しはコンパイラが最適化することもない．このことに着目し，提案手法ではプログラム $p$ を入力 $I$ で実行したときのAPI呼び出しの履歴を取得し，APIの呼び出し順序を実行系列バースマーク，各APIの呼び出し頻度を実行頻度バースマークとして抽出する．提案手法は，機械語プログラムに対して適用でき，ソースコードを用いることなくバースマークを抽出できる．

本章では，まず議論の明確化のために，ソフトウェアのコピーとは何か，バースマークとは何かを定義し，理想的なバースマークの性質について述べる．次に，提案手法のキーアイデアと，2種類のバースマークの定義とそれぞれの類似度について述べる．その後，Windows環境を対象に，実際にソフトウェアの実行時のAPI呼び出しを観測し，その情報から2種類の提案バースマークを抽出する方法を述べる．

### 4.2 準備

#### 4.2.1 コピー関係

議論を明確にするため，まず最初にソフトウェアのコピー関係を定義する．

定義 1 (コピー関係)  $Prog$  を与えられたプログラムの集合とする。そして、 $\equiv_{cp}$  を  $Prog$  上の以下のような同値関係とする。  $p, q \in Prog$  に対し、 $q$  が  $p$  のコピーであるならば、そのときに限り  $p \equiv_{cp} q$  が成り立つ。このとき、 $\equiv_{cp}$  をコピー関係と呼ぶ。

$q$  が  $p$  のコピーであるかどうかの基準は絶対的なものではなく、状況に依存する。例えば、以下の基準は一般的なプログラムとしてはコピーであるといえる。

(a)  $q$  は  $p$  の完全な複製である

(b)  $q$  は  $p$  のソースコード中に現れる全てのシンボル名を変更したものである

(c)  $q$  は  $p$  のソースコードからコメント行を全て削除したものである

ここでは、混乱を避けるため、 $\equiv_{cp}$  はユーザにより与えられるものとする。また、 $\equiv_{cp}$  は同値関係であるから、以下の命題を満たす。いずれもコピーの概念と直感的に一致する。

命題 1  $p, q, r \in Prog$  ならば以下の性質を満たす。

(反射律)  $p \equiv_{cp} p$

(対称律)  $p \equiv_{cp} q \Rightarrow q \equiv_{cp} p$

(推移律)  $(p \equiv_{cp} q) \wedge (q \equiv_{cp} r) \Rightarrow p \equiv_{cp} r$

次に、 $q$  が  $p$  のコピーである場合の  $p$  と  $q$  の外面的な振る舞いは同じであるから、以下が成り立つ。

命題 2  $Spec(p)$  を  $p$  の (外部) 仕様とする。このとき、 $p \equiv_{cp} q \Rightarrow Spec(p) = Spec(q)$  を満たす。

この命題の逆は必ずしも成り立たない。なぜなら、全く独立に同じ仕様のプログラムが実装されることがあり得るためである。

#### 4.2.2 ソフトウェアバースマーク

与えられた  $Prog$  と  $\equiv_{cp}$  に対し，動的バースマークの概念を定義する．この定義は，Myles らによって与えられた [42] ．

定義 2 (動的バースマーク)  $p, q$  を与えられたプログラム， $I$  を与えられた入力とし， $\equiv_{cp}$  を与えられたコピー関係とする． $f(p, I)$  を  $p$  とその入力  $I$  からある方法  $f$  により抽出された特徴の集合とする．このとき，以下の条件を満たすならば， $f(p, I)$  を  $p$  の動的バースマークであるという (図 3) ．

条件 1  $f(p, I)$  はプログラム  $p$  と  $p$  に対する入力  $I$  のみから得られる

条件 2  $p \equiv_{cp} q \Rightarrow f(p, I) = f(q, I)$

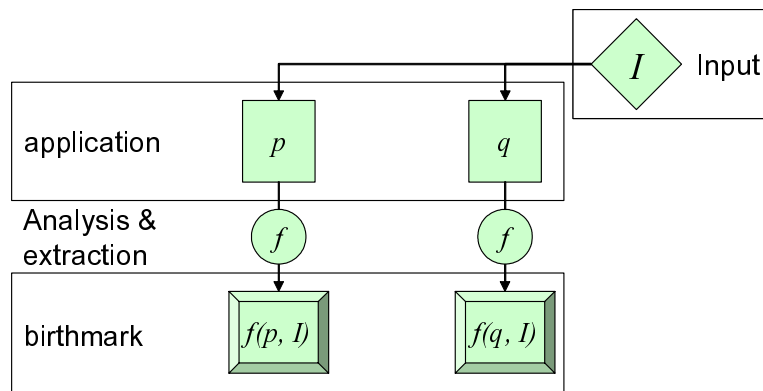


図 3 動的バースマーク

条件 1 は，バースマークはプログラムの付加的な情報ではなく， $p$  の実行に必要な情報であることを示す．すなわち，バースマークは電子透かしのように付加的な情報を必要としないことを表す．条件 2 はコピーされたプログラムからは同じバースマークが得られることを示す．対偶から，もしバースマーク  $f(p, I)$  と  $f(q, I)$  が異なっていれば， $p \not\equiv_{cp} q$  を満たす．これにより， $q$  は  $p$  のコピーではないことが保証される．

### 4.2.3 バースマークの満たすべき性質

バースマークは以下の性質を満たすことが望まれる（図4）。

保存性 (Preservation)  $p$  から任意の等価変換により得られた  $p'$  に対して,  $f(p, I) = f(p', I)$  を満たす

弁別性 (Distinction)  $Spec(p) = Spec(q)$  となる  $p$  と  $q$  に対し, それらが全く独立に実装された場合,  $f(p, I) \neq f(q, I)$  となる

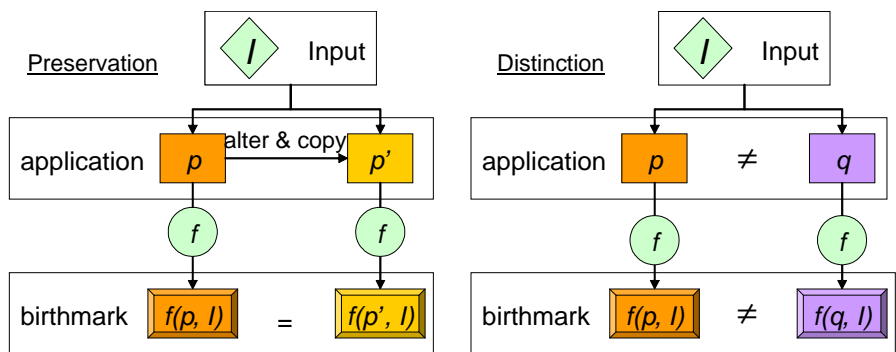


図4 保存性と弁別性

保存性はプログラム変換によりバースマークが変化しないことを表す。攻撃者は盗用の事実を隠すために、元のプログラムを等価な別のプログラムに変換することによりバースマークを改ざんする可能性がある。このような変換を行うための手法には、難読化 [25, 23, 43, 44] や最適化があり、バースマークはこれらの攻撃によっても変化しないことが望ましい。一方、弁別性は全く独立に実装されたプログラム  $p, q$  を正しく区別できることを表す。一般的にプログラムがある程度の規模であれば、プログラムの詳細な部分まで一致することは非常に稀である。しかしながら、たとえ  $p$  と  $q$  が全く独立に実装されていても、プログラムの規模が非常に小さい場合、この性質を満たさない場合がある。一般的にすべてのプログラムに対して、保存性・弁別性を完全に満たすバースマークを設計することは難しい。従って、実用上はユーザの判断により性質の強度を適宜決定する必要がある。

### 4.3 キーアイデア

まず、ソフトウェア内のどのような情報をバースマークとして利用すべきかを考える。例えば、ソフトウェアの「バイナリコード中の XOR 命令の数」は、一見そのソフトウェアのユニークな性質と考えられる。しかし、ソフトウェアの盗用者は、XOR 命令をいくつかの AND 命令と OR 命令に書き換えることで、オリジナルソフトウェアの動作を保存しつつ、XOR 命令を消し去ることが出来る。従って、この情報はバースマークとしては攻撃に弱く (fragile) 実用的ではない。そこで、より改ざんされにくい情報として、我々はソフトウェアの API 呼び出しに着目する。

OS 上で動作するソフトウェアは、API(Application Program Interface) を用いることで、OS の様々な機能を利用することができる。OS が API によって提供する機能としては、ファイル入出力機能、ウィンドウシステムなどのユーザインターフェース機能、セマフォやクリティカルセクションなどの同期機能などがある。

高度な OS は、システム保護の観点から、システムリソースへのアクセスを API を通じてしか許可しない。例えば、ファイル操作を行う場合には、ソフトウェアが直接ファイルシステムを操作することは許可せず、API による操作のみ許可する。このような API による命令は、他の等価な命令によって置き換えることができない。

提案手法では、2 種類のバースマークを提案する。第 1 の実行系列バースマークは、ソフトウェアの API 呼び出し順序に基づくバースマークである。第 2 の実行頻度バースマークは、API それぞれの呼び出し回数に基づくバースマークである。

本論文で特に対象とするソフトウェアは、Windows 環境で動作する、API 呼び出しをある程度利用している中規模以上のソフトウェアとする。API の利用が著しく少ないソフトウェアや、小規模なソフトウェアは対象外である。



## 4.4 提案バースマーク

### 4.4.1 実行系列バースマーク

$p$  を与えられたプログラム,  $I$  を与えられた入力とし,  $I$  によって  $p$  を実行した場合の API 呼び出し順序について考える.  $p$  を改変し, オリジナルの動作を維持しつつ API の呼び出し順序を改変することは困難である. また, プログラム  $p$  と  $q$  が同様の API 群を利用している場合でも, API の呼び出し順序まで同様になることは稀である. このことから, API の呼び出し順序をバースマークとして利用する.

**定義 3 (実行系列バースマーク)**  $p$  を与えられたプログラム,  $I$  を与えられた入力,  $W$  を与えられた API 関数の集合とする.  $(w_1, w_2, \dots, w_n)$  を,  $p$  を入力  $I$  に基づいて実行させたときに呼び出される関数の (順序付きの) 系列とする. このとき,  $w_i (1 \leq i \leq n)$  のうち,  $W$  に属さないものを消去して得られる系列を,  $p$  の  $I$  に基づく実行系列バースマークと呼び,  $EXESEQ(p, I)$  と表記する.

### 4.4.2 実行頻度バースマーク

攻撃者がソフトウェアの意味解析を行った上で API の呼び出し順序を変更したとしても, ある単位時間および単位機能の API 呼び出し頻度は, ほぼ同様となる. このことから, API の呼び出し頻度をバースマークとして利用する.

**定義 4 (実行頻度バースマーク)**  $p$  を与えられたプログラム,  $I$  を与えられた  $p$  に対する入力,  $(w_1, w_2, \dots, w_n)$  を  $p$  の実行系列バースマーク ( $EXESEQ(p, I)$ ) とする.  $(w'_1, w'_2, \dots, w'_m)$  を  $EXESEQ(p, I)$  から関数名の重複をなくし, 与えられた特定の順番に並べた系列とする.  $k_i (1 \leq i \leq m)$  を  $w'_i$  の関数名,  $a_i$  を  $EXESEQ(p, I)$  中に現れる  $k_i$  の回数とする. このとき,  $((k_1, a_1), (k_2, a_2), \dots, (k_m, a_m))$  を  $p$  の  $I$  に基づく実行頻度バースマークと呼び,  $EXEFREQ(p, I)$  と表記する.

## 4.5 バースマークの類似度

プログラム  $p, q$  とそれらに対する入力  $I$  に対し, それぞれバースマーク  $f(p, I) = (p_1, p_2, \dots, p_n)$ ,  $f(q, I) = (q_1, q_2, \dots, q_n)$  が得られたとする. このとき, 全ての  $i$  に対して  $p_i = q_i$  である場合,  $f(p, I)$  と  $f(q, I)$  は同一のバースマークであるといい,  $f(p, I) = f(q, I)$  と書く. この場合, 1 つでも  $p_i \neq q_i$  の組があれば, ほかの全てのペアが等しくても,  $f(p, I) \neq f(q, I)$  となってしまう. よって, 2 つのバースマークが非常に似ているにもかかわらず,  $p \neq_{cp} q$  と結論づけてしまう. このように, 全く同じかそうでないかでバースマークを比較すると, 手法そのものが敏感になりすぎるため実用的ではない.

この問題に対処するため, 我々は提案バースマークの類似度を定義する. 実行系列バースマークでは, 類似度として相互に含まれる部分系列の全体に対する比率を用いる. また, 実行頻度バースマークでは, それぞれのバースマークをベクトルとし, ベクトル間の角度を類似度として用いる.

定義 5 (実行系列バースマークの類似度)  $p, q$  を与えられたプログラム,  $I$  を与えられた  $p, q$  に対する入力とする. さらに,  $p, q$  の実行系列バースマークをそれぞれ  $EXESEQ(p, I) = \rho_p = (wp_1, wp_2, \dots, wp_n)$ ,  $EXESEQ(q, I) = \rho_q = (wq_1, wq_2, \dots, wq_m)$  とする.  $\rho_p$  と  $\rho_q$  が共にある系列  $\rho = (r_1, r_2, \dots, r_k)$  を含む場合, すなわち,  $(r_1 = wp_i = wq_j), (r_2 = wp_{i+1} = wq_{j+1}), \dots, (r_k = wp_{i+(k-1)} = wq_{j+(k-1)})$  なる  $i$  と  $j$  が存在する場合,  $\rho$  を  $EXESEQ(p, I)$  と  $EXESEQ(q, I)$  の一致列と呼ぶ.  $\rho_p$  と  $\rho_q$  の一致列のうち, 最も長い一致列  $\rho$  を最長一致列と呼ぶ. 最長一致列が  $\rho = (r_1, r_2, \dots, r_k)$  のとき,  $k$  を最長一致列長と呼ぶ. このとき,  $EXESEQ(p, I)$  と  $EXESEQ(q, I)$  の類似度を以下の式で求める.

$$\frac{2k}{m+n}$$

この類似度は,  $0 \leq k \leq n$ ,  $0 \leq k \leq m$  であるため, 0.0 から 1.0 までの値をとる.

定義 6 (実行頻度バースマークの類似度)  $p, q$  を与えられたプログラム,  $I$  を与えられた  $p, q$  に対する入力とする.  $p, q$  の実行頻度バースマークをそれぞれ

$EXEFREQ(p, I) = ((k_{p_1}, ap_1) \dots (k_{p_n}, ap_n))$  ,  $EXEFREQ(q, I) = ((k_{q_1}, aq_1) \dots (k_{q_n}, aq_n))$   
とする .  $EXEFREQ(p, I)$  ,  $EXEFREQ(q, I)$  から ,  $\vec{v}_p = [ap_1, ap_2, \dots, ap_n]$  ,  $\vec{v}_q = [aq_1, aq_2, \dots, aq_n]$  なるベクトルを構成する . このとき ,  $EXEFREQ(p, I)$  と  $EXEFREQ(q, I)$  の類似度を以下の式で求める .

$$\frac{ap_1aq_1 + ap_2aq_2 + \dots + ap_naq_n}{\sqrt{ap_1^2 + ap_2^2 + \dots + ap_n^2} \sqrt{aq_1^2 + aq_2^2 + \dots + aq_n^2}}$$

これは ,  $\vec{v}_p$  と  $\vec{v}_q$  のなす角度のコサインである . 全ての  $i$  について  $p_i$  と  $q_i$  は正の整数であるため , 類似度は 0.0 から 1.0 までの値をとる .

#### 4.6 実装の概要

ソフトウェア実行時に API 呼び出しを観測し , 提案バースマークを抽出する手法を , MS-Windows 環境下で行う方法を述べる . 実際のソフトウェアの盗用では , 盗用ソフトウェアのソースコードを入手できることは稀であるため , ソースコードを用いることなくバースマークを抽出できることは重要である . 本実装は , ソースコードを必要とせず , 実行ファイルのみから提案バースマークを抽出する .

バースマークを抽出するためには , ソフトウェア実行時の API 呼び出しを観測する必要がある . Windows では , Windows フックと呼ばれる機能で , ユーザーインターフェース操作のためのメッセージ機構の監視 , キーボードおよびマウス入力の監視が可能であるが [45] , 提案バースマーク抽出に必要な詳細情報の収集までは行うことができない . そこで , 以下のような方針を採用する .

Step1: 実行中の対象ソフトウェアに対し , API 呼び出し観測ルーチンを挿入する

Step2: API 呼び出しのための関数ポインタテーブルを変更する

Step3: API 呼び出しを記録する

Step4: オリジナル API を呼び出す

Step5: バースマークを抽出する

## 4.7 実装の詳細

### 4.7.1 Step1: API 呼び出し観測ルーチンの挿入

API 呼び出しを観測するために、実行中の対象ソフトウェアに API 呼び出し観測ルーチンを挿入する必要がある。ここでは、観測ルーチンを DLL (ダイナミックリンクライブラリ) として実装し、この DLL を実行中の対象ソフトウェアに強制的にロードするアプローチを取る (以降、挿入する DLL をパラサイト DLL と呼ぶ)。通常 Windows では、ソフトウェア自ら指定した DLL もしくはシステムが指定した DLL しかロードすることができない。そこで、我々は Windows フックを用いて対象ソフトウェアにパラサイト DLL を強制的にロードさせる手法 [46] を用いる。図 5 に Windows フックの動作を示す。Windows のメッセージ機構は、通常システムからウィンドウにメッセージが直接送られるが (図 5: 破線矢印)、Windows フックを使用することによりメッセージを横取りすることができる (図 5: 実線矢印)。Windows フックでは、メッセージの横取りを実現するために、指定したメッセージ監視関数を含む DLL を実行中のソフトウェアにロードさせる機能を持つ。本実装では、Windows フックの本来の目的であるメッセージの監視機能は使用せず、パラサイト DLL を強制的にロードさせる目的にのみ用いる。

### 4.7.2 Step2: API 関数ポインタテーブルの変更

実行中のソフトウェアが API を呼び出す際の動作は以下の通りである。実行中のソフトウェアは、インポートセクションと呼ばれる領域を持つ。これは、API が実装されている DLL への関数ポインタを含むポインタテーブルである。図 6 の破線矢印で、通常の API 呼び出しを示す。ソフトウェアが API を呼び出すと、呼び出した API に対応する関数ポインタをインポートセクションから得て、API が実行される。従って、インポートセクションを変更すれば、ソフトウェアが API を呼び出したときに実行される関数を変更することができる [46]。DLL はロードされた時点で、その初期化コードが呼び出される。本実装では、パラサイト DLL の初期化コードでインポートセクションを変更する。具体的には、観測したい API

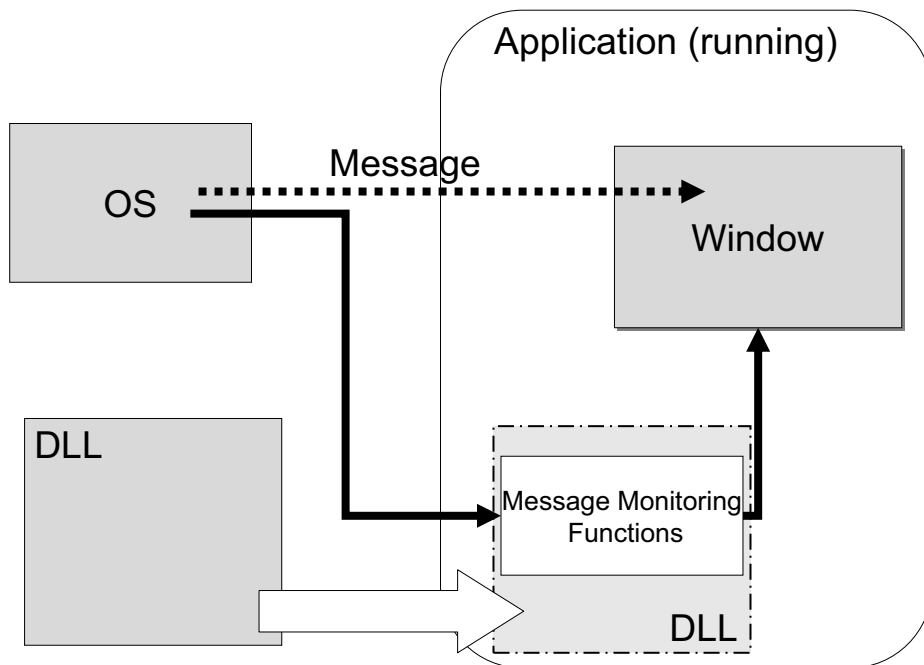


図 5 実行中のソフトウェアに DLL を挿入

に対応するインポートセクションのエントリを、パラサイト DLL に含まれる API 観測ルーチンへのポインタに書き換える。

#### 4.7.3 Step3: API 呼び出しの記録

Step2 でインポートセクションを書き換えたことにより、実行中のソフトウェアが API を呼び出すと、本来呼び出される API 関数の代わりにパラサイト DLL の API 観測ルーチン（ラッパー関数）が呼び出される（図 6：実線矢印）。ラッパー関数では、以下の情報をログファイルに書き出す。

- 呼び出した API の種類（名前）
- API を呼び出した時間

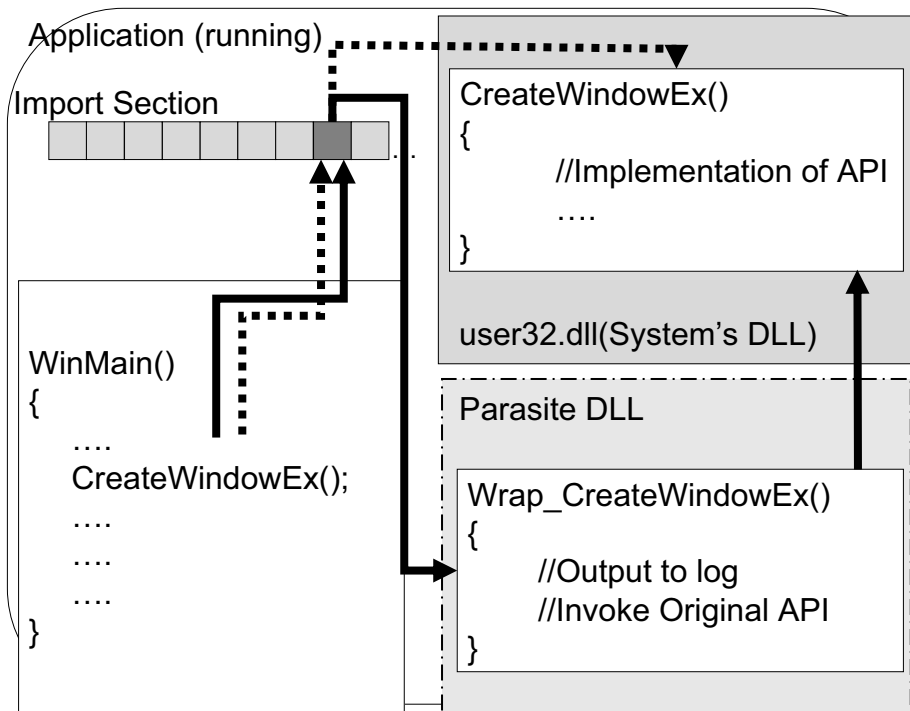


図 6 API呼び出しの観測

- APIを呼び出したスレッドID

時間とスレッドIDは実行系列バースマークを抽出する際に用いられる。

#### 4.7.4 Step4: オリジナルAPIの呼び出し

対象ソフトウェアにパラサイトDLLを挿入しても、ソフトウェアのオリジナルの動作を維持しなければならない。そこで、パラサイトDLLのラッパー関数は、Step3のあとにオリジナルのAPI関数を呼び出し、その戻り値をラッパー関数の戻り値とする。図2の実線矢印で、対象ソフトウェアにパラサイトDLLが挿入された状態でAPIを呼び出した際の動作を示す。これにより、パラサイトDLLを挿入しても、ソフトウェアのオリジナルの動作は維持される。

#### 4.7.5 Step5: バースマークの抽出

パラサイト DLL を挿入したソフトウェアを動作させることで、API 呼び出しを観測し、情報を収集することができる。盗用が疑われる機能があれば、実際にユーザがその機能を実行することで、該当箇所のバースマークを得ることができる。得られたデータを解析し、API 呼び出し記録をスレッド ID ごとにまとめ、時間順にソートすることで実行系列バースマークを得る。また、API 呼び出しの出現回数から実行頻度バースマークを得る。

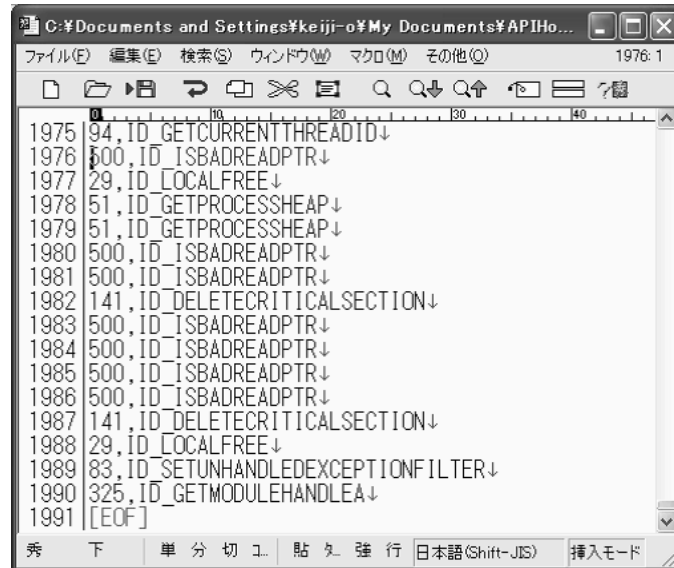
一般的に、マルチスレッドプログラムの実行系列をスレッドを区別せずに抽出した場合、複数のスレッドがどの順序で実行されるかは定まらないため、得られる実行系列も一意に定まらない。しかし、提案手法はスレッド ID により API を呼び出したスレッドを識別することが出来、その呼び出し系列をスレッド ID ごとに分類している。そのため、マルチスレッドに起因する実行系列の乱れは発生しない。

### 4.8 バースマーク抽出ツール – K2 Birthmark Tool Kit (K2BTK)

以上の実装アプローチに基づき、我々はバースマーク抽出ツール K2 Birthmark Tool Kit (K2BTK)[47] を実装した。K2BTK は、任意の Windows アプリケーションから、そのソースコードを用いることなく、実行系列バースマーク、実行頻度バースマークを抽出できる。また、実行系列バースマーク同士、実行頻度バースマーク同士の類似度を求めることができる。実装言語は、C++ および Delphi で約 11,000 行である。

MS-Windows はその基本 API として、Microsoft Platform SDK[48] に含まれる基本ヘッダファイル winbase.h で宣言された 613 種類を備えている。このうち、K2BTK では、以下の 14 種類の API を除いた 599 種類の API を観測可能である：EnterCriticalSection(), LeaveCriticalSection(), TLSGetValue(), RestoreLastError(), LocalLock(), LocalUnlock(), HeapAlloc(), HeapFree(), WriteProfileStringA(), LoadLibraryA(), LoadLibraryW(), LoadLibraryExA(), LoadLibraryExW(), GetProcAddress()。これら 14 種類の API は、ほとんどのアプリケーションにお

いて呼び出し回数が膨大になりすぎる API や、K2BTK の動作そのものに必要な API であり、現在の実装から意図的に除外されている。図 7 に K2BTK によって得られた API 呼び出し系列の例を示す。



```
C:\Documents and Settings\keiji-o\My Documents\APIHo...
ファイル(F) 編集(E) 検索(S) ウィンドウ(W) マクロ(M) その他(O) 1976:1
0 10 20 30 40
1975 94, ID_GETCURRENTTHREADID↓
1976 500, ID_ISBADREADPTR↓
1977 29, ID_LOCALFREE↓
1978 51, ID_GETPROCESSHEAP↓
1979 51, ID_GETPROCESSHEAP↓
1980 500, ID_ISBADREADPTR↓
1981 500, ID_ISBADREADPTR↓
1982 141, ID_DELETECRITICALSECTION↓
1983 500, ID_ISBADREADPTR↓
1984 500, ID_ISBADREADPTR↓
1985 500, ID_ISBADREADPTR↓
1986 500, ID_ISBADREADPTR↓
1987 141, ID_DELETECRITICALSECTION↓
1988 29, ID_LOCALFREE↓
1989 83, ID_SETUNHANDLEDEXCEPTIONFILTER↓
1990 325, ID_GETMODULEHANDLEA↓
1991 [EOF]
秀 下 単 分 切 工 貼 久 強 行 日本語(Shift-JIS) 挿入モード
```

図 7 導出された API 系列の例



## 5. 保存性・弁別性の評価

提案手法の有効性を評価するため，2種類の実験を行った．実験の目的は，提案バースマークの保存性と弁別性を，同じ用途の実用ソフトウェア群を対象に評価すること，および，コンパイラの最適化に対する提案バースマークの耐性を評価することである．

実験環境としては，ハードウェアとして DELL LATITUDE D600 ( Intel Mobile PentiumM 1.6GHz, 1GB RAM ) を使用した．OS は Microsoft Windows XP Professional SP1 である．

### 5.1 実験1 バースマークの保存性，弁別性の評価

本実験では，提案バースマークの保存性と弁別性の評価を行う．実験のために，同用途のソフトウェアを5つ用意し，提案バースマークによってそれらの類似度を評価した．

実験に使用したソフトウェアおよびその作者を表1に示す．これらは全てMP3オーディオファイルのメタ情報タグを編集するソフトウェアである．Super Tag Editor 改 ( STE 改またはSTE-Ext ) [49] は，Super Tag Editor 2.00b7 ( STE ) [50] のソースコードを利用して作成された改変バージョンであり，作者がオリジナルのSTEを合法的に引き継いで作成された．つゆたく ( Tuyutag ) [51]，Teatime[52]，Mp3tag[53] はそれぞれSTEとは全く独立に作成されたソフトウェアである．

まず，参考実験として，5つの実行ファイルそのものの比較を行った．この比較のため，バイナリ差分作成ツールWDiff[54]を用いてバイナリデータの差分を抽出し ( WDiffの厳密差分モード2を使用 ) ，ファイルサイズに占める差分への割合を求めた．表2に各ソフトウェアのペアを比較した結果を示す．表中で，ファイルサイズ ( File size ) はそれぞれのソフトウェアの実行ファイルの大きさ ( 単位：バイト ) ，差分サイズ ( Diff size ) は各ソフトウェアの実行ファイルを更新するために必要な差分データの大きさ ( 単位：バイト ) である．また，割合 ( Percent ) は比較する2つの実行ファイルの大きさのうち，大きなほうに占める差分サイズの割合である．この結果から，実行ファイルをデータとして比較した場合には，

表 1 実験 1 で使用したソフトウェア

Applications	Authors
Super Tag Editor 2.00b7 (STE)	MERCURY
Super Tag Editor Extended Rev.32 (STE-Ext)	haseta
Tuyutag 2.02	P's Soft
Teatime 2.525	Toru Kuroda
Mp3tag 2.2.6.0	Florian Heidenreich

表 2 実行ファイルの差分サイズによる類似性評価

	STE-Ext		Tuyutag		TeaTime		Mp3tag		File size
	Diff size	Percent	Diff size	Percent	Diff size	Percent	Diff size	Percent	
STE	706,189	73.1%	903,345	94.4%	954,839	94.9%	1,626,527	89.8%	565,248
STE-Ext	-	-	902,370	93.3%	953,882	94.8%	1,612,652	89.1%	966,656
Tuyutag	-	-	-	-	771,803	76.7%	1,674,465	92.5%	956,928
TeaTime	-	-	-	-	-	-	1,674,023	92.5%	1,006,080
Mp3tag	-	-	-	-	-	-	-	-	1,810,432

それぞれのソフトウェアは全く類似性を示さないことがわかる。

次に、提案バースマークによる比較を行う。実験では、各ソフトウェアを起動したのち即終了し、起動から終了までに呼び出される API の呼び出し履歴から、2種類の提案バースマークを抽出する。そして、各ソフトウェアのバースマークをそれぞれ比較した。API呼び出し履歴の記録、バースマークの抽出およびバースマークの比較には K2BTK を用いた。K2BTK の性能の目安として、例えば STE と STE 改のバースマークの比較に必要な時間は、実行系列バースマークの比較に約 13 秒、実行頻度バースマークの比較に約 0.05 秒をそれぞれ要した。

表 3 に実行系列バースマークの結果を示す。この結果から、実行系列バースマークの類似度は、STE と STE 改との間で非常に高く、STE と STE 改以外との間ではいずれも低いことがわかる。

表 3 実験 1 結果: 実行系列バースマーク (EXESEQ) による類似度評価

	STE	STE-Ext	Tuyutag	TeaTime	Mp3tag
STE	1.0000	0.9566	0.0107	0.0096	0.0019
STE-Ext	-	1.0000	0.0106	0.0095	0.0019
Tuyutag	-	-	1.0000	0.0115	0.0011
TeaTime	-	-	-	1.0000	0.0008
Mp3tag	-	-	-	-	1.0000

表 4 実験 1 結果: 実行頻度バースマーク (EXEFREQ) による類似度評価

	STE	STE Ext	Tuyutag	TeaTime	Mp3tag
STE	1.0000	0.9997	0.3855	0.2891	0.0182
STE-Ext	-	1.0000	0.3960	0.3074	0.0191
Tuyutag	-	-	1.0000	0.7105	0.1157
TeaTime	-	-	-	1.0000	0.0837
Mp3tag	-	-	-	-	1.0000

次に、図 8 に各ソフトウェアの実行頻度バースマークを示す。図中で、 $x$  軸は API の種類、 $y$  軸は各 API の呼び出し回数の対数、 $z$  軸はソフトウェアの種類である。図 8 からわかるように、STE と STE 改の実行頻度バースマークは非常に類似している。各ソフトウェア間の実行頻度バースマークによる類似度を表 4 に示す。この結果から、実行頻度バースマークの類似度は STE と STE 改との間で非常に高く、完全一致を示す 1.0 に極めて近いが、それ以外のソフトウェア間ではいずれも低いことがわかる。<sup>1</sup>

<sup>1</sup> 表 4 においてつゆたぐと TeaTime に 0.7105 の類似度が確認できるが、実行頻度はコサイン類似度によって比較されるため、この値が高い類似度を表しているわけではない(図 8 参照)。実際  $\arccos(0.7105) \approx \pi/4$  であり、ベクトルのなす角としては完全一致の場合 ( $= 0$ ) と直行する場合 ( $= \pi/2$ ) の中間値となる。

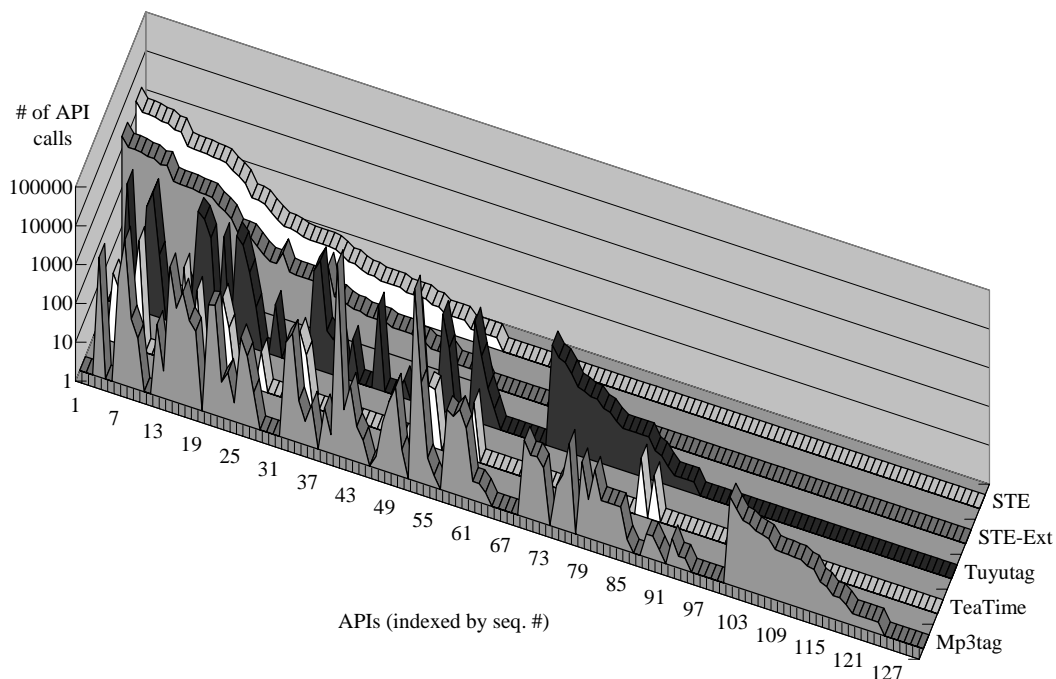


図 8 実行頻度バースマークのグラフ

このように，提案する実行系列バースマークと実行頻度バースマークは，同用途のソフトウェア群について，オリジナルとその改変バージョンである STE と STE 改との間で高い類似度を示した．一方，全く独立に実装されたソフトウェア間についてはいずれも低い類似度を示した．

またこの他にも，ワードプロセッサである Word2000[55] と一太郎 13[56]，一太郎 2004[57] を相互に比較した．一太郎 13 と一太郎 2004 を比較した結果，EXESEQ，EXEFREQ の類似度はそれぞれ，0.855，0.999 となり，非常に高い類似性を示した．また，Word2000 と一太郎 13，Word と一太郎 2004 の比較においては，どちらも実行系列バースマークの類似度は 0.001 以下，実行頻度バースマークについては 0.133 以下となり，全く異なるバースマークを持つことがわかった．

我々が現在までに行ったケーススタディにおいて，提案バースマークは多くの実用プログラムに対して高い性能を示している．しかしながら，このことが必ず

しも全てのプログラムに対して有効と証明しているわけではない。今後、より多くのケーススタディを通して、提案バースマークの有効性の検証を続けていくことが重要である。

## 5.2 実験2 コンパイラの最適化に対するバースマークの耐性

実験2では、プログラム最適化に対する、提案バースマークの耐性を評価する。具体的には、同一のソースコードから異なるコンパイラやコンパイルオプションを用いて実行ファイルを作成した場合、提案バースマークがどれだけ改変されるかを調べる。

異なるコンパイラや、異なるコンパイルオプションを用いると、異なる実行ファイルが作成される。例えば、デバッグオプションを指定した場合、デバッグコードやシンボルテーブルなどが実行ファイルに付加される。また、リリースオプションを指定すると、コンパイラはソフトウェアが可能な限り高速に実行されるように、最適化を施す。この最適化技術は日々進歩しており、コンパイラのベンダーによってもノウハウが異なるため、異なるコンパイラが生成する実行ファイルは一般に異なる。Windows環境は多数のコンパイラが存在するため、異なるコンパイラやコンパイルオプションを使って、バースマークを改ざんする攻撃が行われる可能性がある。特に、ソースコードが公開されている GPL ソフトウェアなどでは、このような攻撃を受ける可能性が高い。

実験では、実行ファイルを作成するために、Super Tag Editor 2.01 のソースコードを使用し、表5に示すコンパイラを用いてコンパイルを行った<sup>2</sup>。以降、表中の略称を各コンパイラの呼称として使用する。

各コンパイラでは、デバッグビルド (Debug) とリリースビルド (Release) のためのコンパイルオプションを設定し、実行ファイルを作成した。デバッグビルドは、それぞれのコンパイラでデバッグビルドとしてデフォルトで用意されている設定とした。リリースビルドは、ソフトウェアの実行速度が可能な限り速くなるような設定とした。作成された実行ファイルのサイズは表6のようになった。

---

<sup>2</sup> 実験1と同じバージョンである Super Tag Editor 2.00b7 のソースコードを用いるべきではあるが、2.00b7 のソースコードは提供されていないため、2.01 のソースコードを利用した

表 5 実験 2 で使用したコンパイラ一覧

Compilers	Legend
IntelC++ 8.1.024_ev05(evaluation version)	IntelC
Microsoft VisualC++ 6.0 Professional Edition SP6	VC6
Microsoft VisualC++ 7.0 ( VisualStudio.NET 2002 )	VC7

これら 6 つの実行ファイルを提案バースマークを用いて比較する．なお，実験方法は実験 1 と同様である．

表 6 実行ファイルのサイズ

Compilers and options	File size(bytes)
IntelC Debug	2,281,472
IntelC Release	913,408
VC6 Debug	1,699,960
VC6 Release	581,632
VC7 Debug	2,113,536
VC7 Release	577,536

実験 2 では，観測対象の API 集合  $W$  から，以下の 4 種類を除外した: `IsBadReadPtr()`, `IsBadWritePtr()`, `HeapValidate()`, `WaitForMultipleObjectsEx()`．これらの API の呼び出しは，OS の現状態（メモリ使用量や実行時間，実行中のプロセスなど）に非常に敏感であり，同一のソフトウェアを異なるタイミングで実行した場合でも，呼び出し回数が大きく変動する．そこで，同一ソフトウェアの動的バースマークを同一に正規化するための予備実験を行い，これら 4 種類の API を除外することにした．

表 7 に実行系列バースマークの結果を示す．さらに，表 8 に実行頻度バースマークの結果を示す．

表7から，全ての実行ファイルのペアに対し，実行系列バースマークは0.92以上の高い類似度を達成していることがわかる．特に，リリースビルド同士は，コンパイラが異なっても非常に類似している(0.97以上)．また，実行系列バースマークにおける類似度の軽微な差異は，実行頻度バースマークで完全にマスクされている(表8参照)．このことから，APIの実行順序を多少変更したとしても，実行頻度には影響しないことがわかる．表8の実行頻度バースマークの結果では，全てのペアに対し，0.99以上の高い類似度を達成している．

以上の結果から，異なるコンパイラやコンパイルオプションによる，提案バースマークへの影響は軽微であることが確認できた．

表7 実験2結果: 実行系列バースマーク (EXESEQ) による類似度評価

	IntelC		VC6		VC7	
	Debug	Release	Debug	Release	Debug	Release
IntelC Debug	1.0000	0.9492	0.9243	0.9492	0.9988	0.9492
IntelC Release	-	1.0000	0.9392	0.9698	0.9492	0.9996
VC6 Debug	-	-	1.0000	0.9392	0.9243	0.9392
VC6 Release	-	-	-	1.0000	0.9492	0.9699
VC7 Debug	-	-	-	-	1.0000	0.9492
VC7 Release	-	-	-	-	-	1.0000

表 8 実験 2 結果: 実行頻度バースマーク (EXEFREQ) による類似度評価

	IntelC		VC6		VC7	
	Debug	Release	Debug	Release	Debug	Release
IntelC Debug	1.0000	0.9963	0.9966	0.9963	1.0000	0.9963
IntelC Release	-	1.0000	0.9912	1.0000	0.9963	1.0000
VC6 Debug	-	-	1.0000	0.9912	0.9966	0.9912
VC6 Release	-	-	-	1.0000	0.9963	1.0000
VC7 Debug	-	-	-	-	1.0000	0.9963
VC7 Release	-	-	-	-	-	1.0000



## 6. 攻撃耐性の評価

### 6.1 評価のあらまし

ソフトウェアの盗用が実際に行われた場合、盗用者は盗用の発覚を恐れるため、ソフトウェアに対し何らかの改変による攻撃を加える可能性が高い。実験2でとりあげた、異なるコンパイラやコンパイルオプションを用いてソフトウェアを作成することも、改変による攻撃の1つである。

こうしたバースマークへの攻撃は無数に考えられるが、本章では大きく分けて2種類の攻撃に対して提案手法の耐性を評価する。

#### 1. 既存のプログラム自動変換ツールに対する攻撃耐性

特にバースマークへの攻撃を目的とせず、別の目的のために開発されたプログラム変換ツールに対する攻撃耐性を評価する。上記のようなツールには、難読化ツール・最適化ツールなどがある。このような自動変換ツールは、大量の変更を一度に実施できる一方、複雑な意味解析が必要な変換はできない。また、プログラムのオリジナルの動作は維持される。

#### 2. 盗用者による攻撃に対する攻撃耐性

特にバースマークへの攻撃を目的とした、人手による攻撃に対する耐性を評価する。盗用者はバースマーク抽出ツールを使用でき、攻撃結果が確認できることとする。人手による攻撃では、複雑な意味解析によるプログラムの変換が可能である。また、オリジナルの動作を変更するような攻撃をすることもできる。

この評価は、下記の順序で行う。まず盗用者の能力を明らかにする。次に、盗用者が実行可能な攻撃を、API呼び出しの追加・削除・置換・順序入替の4種類に分類し、それぞれ攻撃のコストと攻撃可能回数を検討する。その結果から、どの攻撃がバースマークに対して脅威となるかを検討する。そして、提案バースマークがそれぞれの攻撃によってどのような影響を受けるかを評価する。

## 6.2 既存のプログラム自動変換ツールに対する攻撃耐性

まず、既存のプログラム自動変換ツールに対する攻撃耐性を検討する。プログラム自動変換ツールの特徴を、下記のように整理する。

1. 既存のツールは、特にバースマークへの攻撃を目的としていない。目的は、ツールによって異なる（難読化ツールの場合、ソフトウェアの難読化が目的）
2. 大量の変更を一度に実施できる
3. 複雑な意味解析が必要な変換はできない
4. プログラムのオリジナルの動作は維持される

ここでは、難読化ツールの例を用いて攻撃耐性を評価する。難読化 [25] とは、ソースコードやオブジェクトコードを理解しにくいものに等価変換することにより、ソフトウェアの解析や知的財産の盗用を防ぐための技術である。難読化は本来ソフトウェアの保護を行うものであるが、バースマークを改変する攻撃手法としても用いることができる。難読化ツール [23, 43] は、入力として与えられるソフトウェア（ソースコードあるいは機械語コード）を静的に解析して得られる情報を用いてソフトウェアを変換する。ただし、難読化対象のソフトウェアに外部ライブラリの呼び出しが含まれる場合、外部ライブラリの動作を考慮した変換は行われぬ。従って、一般に難読化ツールは外部ライブラリの呼び出しを別のものに置き換えたり、呼び出しの順序を変更することはできない。そして、API呼び出しは外部ライブラリ呼び出しである。そのため、提案手法は難読化ツールの影響を受けにくい。

また、難読化ツール以外の最適化ツールなど、多くのプログラム自動変換ツールは、同様に外部ライブラリの呼び出しを変更することはできない。よって、提案手法はプログラム自動変換ツールなどによる影響を受けにくいと考えられる。今後、特にバースマークに対する攻撃を目的とした自動変換ツールが開発される可能性はあるが、少なくとも既存のツールの影響は受けないと考えられる。

## 6.3 盗用者による攻撃に対する攻撃耐性

次に，盗用者のバースマークに対する攻撃について検討する．盗用者による攻撃の特徴を，下記のように整理する．盗用者の能力については，次項で盗用者モデルとして詳細に定義する．以降の盗用者モデル，攻撃の最小単位，盗用者の攻撃，攻撃のコストと攻撃可能回数は，森山らの議論 [58] を整理・修正したものである．

1. バースマークへの攻撃を目的としている．盗用者は，バースマーク抽出ツールを使い，攻撃結果を参照できる
2. 手作業による攻撃では，基本的に大量の攻撃はできない．ツールを使えば大量の攻撃が可能な場合もある
3. 複雑な意味解析が必要なプログラム変換も可能
4. 機能そのものの改変による攻撃も可能

### 6.3.1 盗用者モデル

まず，盗用者の能力について下記のモデルを定義する．盗用者の目的は，盗用の事実が発覚しないように盗用を行うことである．盗用者は，ソフトウェア開発とそれに関連する分野のエキスパートであるが，有限のコストを持つものとする．これを踏まえて，盗用者の能力および行動を以下のように仮定する．

1. 盗用対象ソフトウェア  $P$  のソースコード  $P_s$  の記述を全て参照できる．
2. 攻撃能力は限られており，攻撃にかかるコストは，ふるまいが  $P$  と同じソフトウェアを自作するためにかかるコストよりも有意に低い．
3.  $P$  および  $P$  を盗用して作成されたソフトウェアを  $Q$  として， $P, Q$  の任意の動的バースマーク  $P_b, Q_b$  をそれぞれ求めることができ， $P_b$  と  $Q_b$  の類似度を求めることができる．

4.  $Q$  は公開し,  $Q$  のソースコード  $Q_s$  は公開しない.

1 は,  $S$  が公開されているか, 不正に盗み出されたか, または  $P$  の逆コンパイルが可能であることを示す. 2 は, 攻撃を行うためにはコストをかける必要があり, また, かけられるコストは限られていることを示す. 3 は,  $Q$  の作成時に存在する動的バースマークを用いて  $P_b$  と  $Q_b$  の類似度を求め, 攻撃の指標にできることを示す. 4 は, 盗用発見のために  $Q_s$  が使用できないことを示す.

### 6.3.2 攻撃の最小単位

盗用者にとって, 攻撃とはソースコードを改変し API 呼び出し系列を変更することである. 提案手法では, API 呼び出し頻度も用いるが, 呼び出し頻度は呼び出し系列から得ることができる. ここで, API 呼び出し系列を文字列としてとらえると, API 呼び出し系列を変化させる攻撃の最小単位は, 文字列の基本的操作である 1 文字の削除・挿入・置換の 3 種類と考えることができる. 以降, 1 回の攻撃は, API 呼び出し系列に対する 1 回の基本操作に相当するものとする.

### 6.3.3 盗用者の攻撃

盗用者の攻撃として, 表 9 の 4 種類を考える. これらの攻撃は, API 呼び出し系列に対する攻撃として, 6.3.2 で述べた攻撃の最小単位に基づいて考えることができる.

それぞれの攻撃の詳細は以下の通りである.

**API 関数削除攻撃** とは, API 関数の機能をユーザ関数として実装することにより, API 関数呼び出しを消去する攻撃である. 方法としては, 自作攻撃と複製攻撃が挙げられる. 自作攻撃は, API 関数が持つ機能を盗用者自ら実装することで, API 呼び出し履歴を変化させる攻撃である. 複製攻撃は, API 関数の実装を丸ごと複製し利用することで, API 呼び出し履歴を変化させる攻撃である.

表 9 攻撃の分類

基本操作	攻撃名	詳細分類
削除	API 関数削除攻撃	自作攻撃 複製攻撃
挿入	API 関数挿入攻撃	
置換	API 関数置換攻撃	
(2 回)	API 関数順序入替攻撃	

API 関数挿入攻撃 は、API 関数を冗長に呼び出し、API 呼び出し履歴を変化させる攻撃である。

API 関数置換攻撃 は、ある機能を実現している API 関数を、同等機能が実現可能な別の API 関数の組に置き換え、履歴を変化させる攻撃である。自作による削除攻撃の一種と考えることができる。

API 関数順序入替攻撃 は、2 回の基本操作を組み合わせたもので、呼び出し順序が処理に影響のない箇所を探し、順序を入れ替える攻撃である。

#### 6.3.4 攻撃のコストと攻撃可能回数

次に、各攻撃にかかわるコストと攻撃可能回数を検討する。コストは、準備時コストと変更時コストの 2 種類を検討する。準備時コストとは、ソースコードの変更を実施する前に必要な作業のコストである。変更時コストとは、ソースコードの変更作業そのものに必要なコストである。

API 関数削除攻撃 では、自作攻撃の場合、まず準備時コストとして、API 関数の処理内容を解析し、同等の処理を実装する必要があり、準備時コストは高い。変更する際には、API 呼び出しを機械的に置き換えるだけでよく、変更時コストは低い。一度代替関数を用意すればすべての API 呼び出しを置き換えることができるため、攻撃可能な回数は大きくなるが、攻撃対象に

もともと含まれる API 呼び出しの数に制限されるため、中程度とする。複製攻撃の場合、API 関数を移植する作業が必要であるが、自作攻撃と比較して実装の必要はなく、準備時コストは低い。変更時コスト、攻撃可能回数は自作攻撃と同様となる。

API 関数挿入攻撃 では、API 呼び出しを成立させるために、引数の内容などの知識を習得する必要があるが、準備時コストとしては低い。変更時コストは、基本的には機械的な作業により変更が可能と考えられるため、低い。API 呼び出しの挿入は、処理に影響を与えない限り無限に実施できるため、攻撃可能回数は大きくなる。

API 関数置換攻撃 は、自作関数による削除攻撃で、自作関数に API 呼び出しが含まれるものと考えられることができる。このため、変更時コスト、準備時コスト、攻撃回数は自作関数による削除攻撃と同様となる。

API 関数順序入替攻撃 については、呼び出し順序を入れ替えようとする 2 つの API 関数 a,b について考える。まず、準備時コストとしては、対象 API 関数 a,b についての知識の習得が必要となる。しかしコードの実装等は必要ないため、準備時コストは中程度とする。変更時コストとしては、a,b 相互の依存関係、周囲の処理との依存関係を調査する必要がある。しかも、変更箇所ごとに個別の検討が必要となり、変更しても問題ないことが自明な箇所もあれば、詳細に検討しなければ変更できない箇所もあり、平均すると変更時コストは中程度と考えられる。攻撃回数は、そもそも順序を変更できない箇所が多数存在すると考えられるため、小さくなる。

これらの攻撃を行うためにかかるコストをまとめたものが表 10 および表 11 である。

なお、盗用者の攻撃として、ソフトウェアの機能を維持しつつ API 呼び出し系列を変更しようとする攻撃と、ソフトウェアの機能そのものを変更し API 呼び出し系列を変更しようとする攻撃の 2 種類が考えられる。しかし、バースマークを変更するためにダミーの API 呼び出しを追加する操作と、機能を追加する操作は、両方とも API 関数挿入攻撃と考えることができる。機能の削除も同様であ

表 10 攻撃コストの内容

攻撃	準備コストの内容	変更コストの内容
削除攻撃（自作）	API 処理内容の解析，実装	エディタ等で置換
削除攻撃（複製）	コードの準備	エディタ等で置換
挿入攻撃	API 処理内容の解析	手動，エディタ等で置換
置換攻撃	API 処理内容の解析	手動，エディタ等で置換
順序入替攻撃	API 処理内容の解析，	処理依存関係の解析，手動

表 11 攻撃コスト

攻撃	準備時 コスト	変更時 コスト	攻撃 回数	その他
削除攻撃（自作）	高	低	中	
削除攻撃（複製）	低	低	中	API のソースコードは 通常入手困難
挿入攻撃	低	低	大	
置換攻撃	高	低	中	
順序入替攻撃	中	中	小	

る．従って，機能を維持しながら行う攻撃と，機能を改変して行う攻撃を区別する必要はない．

### 6.3.5 提案手法の攻撃耐性

以上のコストと攻撃可能回数から，バースマークを設計する際にどの攻撃を考慮すべきかを考える．まず，最も低コストかつ攻撃回数的大きいものは挿入攻撃である．このため，API 呼び出し履歴を用いたバースマークでは，挿入攻撃の影響を低減することが重要となる．コストと攻撃可能回数を考えると，次点は複

製による削除攻撃であるが、これは API のソースコードを入手できることが条件となるため、Windows などの商用 OS では不可能に近い。次に重要なのは、自作関数による挿入攻撃と置換攻撃である。順序入替攻撃は攻撃コストも比較的高く、攻撃可能回数も少ないため、積極的に考慮する必要性は低いと考えられる。

次に、これまでに提案した実行系列バースマークと実行頻度バースマークについて、上記攻撃に対する耐性を検討する。実行系列バースマークでは、類似度として最長一致列を用いるため、削除・挿入・置換・順序入替のいずれに攻撃に対しても、最長一致列が損なわれてしまうため、極めて弱いと言える。実行頻度バースマークでは、順序入替攻撃の影響は受けない。また、削除・置換攻撃についても高い耐性を持つと考えられる。ソフトウェアに含まれる API 呼び出しのほとんどを削除もしくは置換することは、非常にコストが大きく現実的ではない。このため、削除もしくは置換された API 呼び出しの数は、ソフトウェア全体の API 呼び出し回数より十分小さくなると考えられる。その場合、実行頻度バースマークのベクトルの角度には大きな影響を与えないと考えられる。挿入攻撃については、大量の攻撃が行われた場合はベクトルの角度が大きく変わり、類似度が損なわれてしまう。大量の挿入攻撃は非常に容易であるため、挿入攻撃に対しては弱いと言える。上記の議論をまとめたものを表 12 に示す。

表 12 提案手法の攻撃耐性

攻撃	実行系列バースマーク	実行頻度バースマーク
削除攻撃（自作）	弱い	強い
削除攻撃（複製）	弱い	強い
挿入攻撃	弱い	弱い
置換攻撃	弱い	強い
順序入替攻撃	弱い	無効

以上のように、これまでに提案した実行系列バースマークと実行頻度バースマークは、人手による攻撃に対しては脆弱であると言える。特に、大量の攻撃が可能である挿入攻撃に弱く、これに対処することが望まれる。



## 7. バースマーク比較方法の改良

6章では、提案バースマークは人手による攻撃に対して弱いことが分かった。そこで、提案バースマークの比較方法を改良し、攻撃耐性を向上させることを試みる。具体的には、これまで実行系列バースマークの比較には最長一致列を元にした類似度を使用していたが、これを編集距離を用いたものに変更する。本章では、修正した実行系列バースマークの定義、編集距離の定義と計算方法について述べる。そして、修正したバースマークの実験的評価の結果について述べる。

### 7.1 実行系列バースマーク比較方法の改良

6章では、特に実行系列バースマークが攻撃に弱いことを示した。また、実行頻度バースマーク・実行系列バースマーク共に、大量の挿入攻撃に弱いことを示した。そこで、実行系列バースマークの比較方法を改良し、挿入攻撃に対する耐性向上を目指す。具体的には、これまで実行系列バースマークでは、最長一致列を用いた類似度により比較していたが、これを下記の定義に変更する。

定義 7 (実行系列バースマークの一致率)  $p, q$  を与えられたプログラム,  $I$  を与えられた  $p, q$  に対する入力とする。さらに,  $p, q$  の実行系列バースマークをそれぞれ  $EXESEQ(p, I) = (wp_1, wp_2, \dots, wp_n)$ ,  $EXESEQ(q, I) = (wq_1, wq_2, \dots, wq_m)$  とする。  $EXESEQ(p, I)$ ,  $EXESEQ(q, I)$  の各関数を文字としてとらえて求めた編集距離を,  $ED_{pq}$  とする (編集距離の算出方法は 7.2 で述べる)。このとき,  $EXESEQ$  の一致率の以下の式で求める。

$$\frac{\max(m, n) - ED_{pq}}{\min(m, n)}$$

ここで,  $\max(a, b)$  は  $a$  と  $b$  のどちらか大きな方を返し,  $\min(a, b)$  は  $a$  と  $b$  のどちらか小さな方を返す関数とする。ここで,  $ED_{pq} \leq \max(m, n)$  であり, また  $\max(m, n) - ED_{pq} \leq \min(m, n)$  であるため, 一致率は 0.0 ~ 1.0 の値をとる。

ここで, 上記計算では  $EXESEQ(p, I) \neq EXESEQ(q, I)$  であっても 1.0 を返す可能性があり, 類似度と称するのは適さないため, 一致率と呼ぶことにする。

表 13 編集距離算出アルゴリズム (1)

		p	u	z	z	l	e
	0	1	2	3	4	5	6
p							
z							
z							
e							
l							

表 14 編集距離算出アルゴリズム (2)

		p	u	z	z	l	e
	0	1	2	3	4	5	6
p	1						
z	2						
z	3						
e	4						
l	5						

## 7.2 編集距離

編集距離 [59] とは、2つの文字列間の距離を求めるアルゴリズムで、スペルチェック等に利用されるものである。編集距離は、一方の文字列に対して「1文字の置換」「1文字の削除」「1文字の挿入」の3つの操作を何回行えば、他方の文字列に変換できるかを距離として算出する。編集距離の算出アルゴリズムを以下に示す。ここでは例として文字列“puzzle”と“pzzel”間の編集距離を求める。

まず、長さゼロの文字列をそれぞれの文字列に変換するための操作回数を考える。長さゼロの文字列を“p”に変換するためには、1文字の挿入で可能である。つまり変換に必要な操作回数は1となる。同様に、長さゼロの文字列を“pz”に変換するには、2文字の挿入が必要であり、操作回数は2となる。これを続けて、長さゼロの文字列を“puzzle”に変換するためには6文字の挿入が必要となる。以上の操作回数計算を、表13のように表現する。同様に、長さゼロの文字列から“pzzel”に変換する操作回数を、同様に計算し、表14のように表現する。

次に、表中の残りのセルの値を計算する。計算式は表15の通りである。表中で、Diagonalは対象セルの右上セルのコスト、Aboveは上セルのコスト、Leftは左セルのコストである。minは引数のうち最小の値を返す関数である。

以上の式を用いて、全てのセルの値を計算すると図16となる。右下のセルの値3が、“puzzle”から“pzzel”に変換する(もしくは逆)ための最小の操作回数で

表 15 編集距離 セルの値の算出式

Diagonal	Above
Left	上と左の文字が同じ場合： $\min(\text{Diagonal}, \text{Above} + 1, \text{Left} + 1)$ 上と左の文字が異なる場合： $\min(\text{Diagonal} + 1, \text{Above} + 1, \text{Left} + 1)$

あり，2つの文字列間の編集距離となる．

表 16 編集距離算出アルゴリズム (3)

		p	u	z	z	l	e
	0	1	2	3	4	5	6
p	1	0	1	2	3	4	5
z	2	1	1	1	2	3	4
z	3	2	2	1	1	2	3
e	4	3	3	2	2	2	2
l	5	4	4	3	3	2	3

### 7.3 改良した実行系列バースマークの攻撃耐性

比較方法を改良した実行系列バースマークが，6章の4種類の攻撃に対してどの程度耐性を持つのかを評価する．

API 関数削除攻撃 API 関数削除攻撃を，表 17 のように表現する．表中で，攻撃前の API 呼び出し系列が攻撃前，攻撃後の呼び出し系列が攻撃後，攻撃前と攻撃後の系列から，編集距離を求めた結果を記号に示す．記号中で， $\backslash$ ， $|$  はそれぞれ編集距離における削除，挿入，置換の各操作を示す．ここでは番号で示す 1,2,3,7 の API 呼び出しが削除攻撃を受けたとする．

表 17 API 関数削除攻撃の模式図

番号	攻撃前	記号	攻撃後
1	a		
2	a		
3	b		
4	b		b
5	b		b
6	c		c
7	a		

表 18 API 関数挿入攻撃の模式図

番号	攻撃前	記号	攻撃後
1	a		a
2			c
3			c
4	a		a
5	b		b
6			z
7	b		b
8	b		b
9			y
10			y
11			y
12	c		c
13	a		a

表中の記号の数から，この2つの系列間の編集距離は4となる．一致率は  $\frac{7-4}{3} = 1.0$  となり，攻撃の影響を受けないことがわかる．

**API 関数挿入攻撃** API関数挿入攻撃を，表18のように表現する．ここで，2,3,6,9,10,11のAPI呼び出しが削除攻撃を受けたとする．攻撃前と攻撃後の系列から，編集距離を求めた結果を記号に示す．

表中の記号の数から，この2つの系列間の編集距離は6となる．一致率は  $\frac{13-6}{7} = 1.0$  となり，攻撃の影響を受けないことがわかる．

**API 関数置換攻撃** API関数置換攻撃を，表19のように表現する．ここで，全てのbをxyに置き換える攻撃を受けたとする．攻撃前と攻撃後の系列から，編集距離を求めた結果を記号に示す．

表 19 API 関数置換攻撃の模式図

攻撃前	記号	攻撃後
a		a
a		a
b		x
b		y
b		x
		y
		x
		y
c		c
a		a

表 20 API 関数順序入替攻撃の模式図

記号	攻撃前	記号	攻撃後
1	a		a
2	a		
3	b		
4	b		b
5			a
6	b		b
7	c		c
8			b
9	a		a

表中の記号の数から，この2つの系列間の編集距離は6となる．改良した一致率は，計算式から  $\frac{10-6}{7} = 0.57$  となる．従来の類似度は0.43，実行頻度バースマークの類似度は0.24となる．一般的に考えて，従来の類似度では，置換攻撃により最長一致列が分断された場合，類似度が大きく低下するが，改良した一致率はAPI関数置換攻撃に対しても影響を軽減できる．

**API関数順序入替攻撃** API関数置換攻撃を，表20のように表現する．ここで，番号で示す2と3，そして5と6を入れ替える攻撃を受けたとする．

表中の記号の数から，この系列の編集距離は4となる．改良した一致率は  $\frac{7-4}{7} = 0.43$  となる．従来の類似度は0.24，実行頻度バースマークは1.00となる．一般的に考えると，従来の類似度では，順序入替攻撃により最長一致列が分断された場合，類似度が大きく低下するが，改良した一致率はAPI関数順序入替攻撃の影響を軽減できる．

## 7.4 攻撃耐性のまとめ

以上をまとめると、比較方法を改良した実行系列バースマークは、挿入攻撃と削除攻撃の影響を受けない。また、置換攻撃と順序入替攻撃に対しても、最長一致列を使った類似度と比較して、攻撃耐性の向上が期待できる。6.3.5 で述べた通り、API 呼び出し履歴を用いたバースマークへの攻撃としては、挿入攻撃が最も容易かつ攻撃可能回数も多いため、挿入攻撃の影響を受けないことは非常に重要である。

比較方法を改良した実行系列バースマークは、比較するバースマークのどちらかが非常に短い場合（例えば長さが1だった場合など）、全く関連のないソフトウェア同士でも高い一致率となる可能性がある。しかし、提案バースマークは4.3 で述べたように「API 呼び出しをある程度利用している中規模以上のソフトウェア」を対象としており、非常に短いバースマークが得られるような小規模なソフトウェアは対象外であるため、問題はない。

また、改良により挿入攻撃を受けないことから、部分盗用発見を発見できる可能性についても期待できる。例として、あるライブラリの盗用について考える。ライブラリ部分のバースマークを取得し、盗用ソフトウェアのバースマークと比較すれば、高い類似度となることが期待されるためである。

## 7.5 実験的評価

改良した実行系列バースマークの実験的評価の結果について述べる。実験対象のアプリケーションとしては、一太郎 2004[57] を使用した。実験では、1 回目は一太郎 2004 を起動したのち即終了し、起動から終了までに呼び出される API の呼び出し履歴を記録した。2 回目は、一太郎 2004 を起動したのち適当な文書ファイルを開き、終了させるまでの API の呼び出し履歴を記録した。2 回目の操作により、挿入攻撃を受けたものと同等の API 呼び出し履歴が得られると考えられる。それぞれから 3 種類のバースマークを抽出し、類似度および一致率を評価した。

結果を表 21 に示す。結果から、比較方法を改良した実行系列バースマークは高い一致度となり、従来の類似度を使った実行系列バースマークは低い類似度と

表 21 改良実行系列バースマーク 実験的評価の結果

	EXESEQ (オリジナル)	EXEFREQ	EXESEQ (改良)
類似度/一致率	0.4472	0.9999	0.9526

なった。従来の類似度では最長一致列が挿入攻撃により分断され、類似度が低くなったと考えられる。これに対し、改良後は挿入攻撃の影響を大きく軽減していることがわかる。

## 8. おわりに

本論文では、ソフトウェア盗用の発見および立証を効率的に行うための技術として、2種類の動的バースマークを提案した。それぞれ、APIの呼び出し順序に注目した実行系列バースマークと、APIそれぞれの呼び出し回数に注目した実行頻度バースマークである。提案手法は、機械語プログラムに対して利用でき、ソースコードを利用することなくバースマークを抽出できる。そして、Windows環境を対象に、ソフトウェアの実行時のAPI呼び出しを観測し、その情報から2種類の提案バースマークを抽出する方法を示した。

また、提案手法の有効性を評価するために、2種類の実験を行った。実験1では、オリジナルのソフトウェアと派生ソフトウェア、同用途のソフトウェアであるが全く別個に作成された3つのソフトウェア、計5つを対象にし、実行ファイルの類似性および提案バースマークによる類似度を評価した。その結果、提案バースマークは、これらの実用ソフトウェア群に対して十分な保存性・弁別性を有することを確認した。第2の実験では、同一のソースコードを用いて異なるコンパイラやコンパイルオプションによってソフトウェアを作成し、提案手法によるそれぞれの類似度について評価した。その結果、異なるコンパイラやコンパイルオプションによる提案バースマークへの影響は軽微であることを確認した。

さらに、提案バースマークの攻撃耐性について評価した。提案バースマークは、既存の機械的変換ツールに対して耐性を持つことが期待される。しかし、手作業による攻撃に対しては弱いことが分かった。

そこで、より攻撃に強いバースマークとするため、実行系列バースマークの比較方法の改良について述べた。比較方法を改良した実行系列バースマークは、APIの挿入と削除による攻撃を無効にし、攻撃に対する高い耐性が期待できることを示した。また、評価実験により、挿入攻撃に対する耐性を確認した。

今後の課題として、置き換え不可能なAPIのみを観測対象とすることで攻撃耐性を向上させること、改良した実行系列バースマークの広範な実験を行うこと、さらに攻撃耐性の強い類似度算出方法を開発することなどが挙げられる。

バースマークは、ソフトウェアそのものに対して何の変更も加えることなく、特徴となる情報を抽出する。このため、ソフトウェアから複数のバースマークを



抽出することが可能である。多くのバースマークを併用することで、盗用発見の精度向上や、攻撃耐性の向上が可能になると考えられる。また、ウォーターマークと競合することもない。ソフトウェアの盗用に対抗するためには、これらの技術を総合的に使用していく必要があると考えられる。

## 謝辞

研究を遂行するにあたって、多くの方々のご指導ご協力を頂きました。

まず、奈良先端科学技術大学院大学 情報科学研究科 松本 健一 教授に厚く御礼を申し上げます。松本教授には、研究のご指導と助言のみならず、研究者としての姿勢など、多くの点を学ぶことができました。また、先生のご尽力とお人柄により、ソフトウェア工学講座は非常にすばらしい研究環境でした。ありがとうございます。

奈良先端科学技術大学院大学 情報科学研究科 関 浩之 教授には、研究に関して貴重なご指導を頂きました。先生のご指導により、本論文をより有意義なものとすることができました。ありがとうございます。

奈良先端科学技術大学院大学 情報科学研究科 楫 勇一 准教授には、研究に関して有益なアドバイスを頂きました。ありがとうございます。

奈良先端科学技術大学院大学 情報科学研究科 門田 暁人 准教授には、研究のご指導とご助言、また数々のご相談をさせていただきました。先生のご指導・ご協力なくして、社会人学生として入学し研究を進めること、また、本論文を完成することはできませんでした。ありがとうございます。

神戸大学 中村 匡秀 准教授には、多大なご指導と助言を頂きました。先生のご協力なくして、電子情報通信学会への論文投稿を達成することはできませんでした。ありがとうございます。

奈良先端科学技術大学院大学 情報科学研究科 玉田 春昭 特任助教には、多大なご指導とご助言をいただきました。研究のバックグラウンドに関する部分の構成や、電子情報通信学会への論文投稿と修正は、先生のご協力なしでは達成することはできませんでした。ありがとうございます。

奈良先端科学技術大学院大学 情報科学研究科 大平 雅雄 助教には、発表練習等で多大なご指導とご助言を頂きました。ありがとうございます。

奈良先端科学技術大学院大学 情報科学研究科 角田 雅照 特任助教には、研究や学生生活にわたって多くのご助言やご協力をいただきました。ありがとうございます。

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学講座 山内 寛

己さん，柿元 健さん，栗山 進さんには，研究や私事にわたり多くのご助言やご協力をいただきました．ありがとうございます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学講座 亀井 靖高さんには，研究や学生生活にわたって多くのご協力をいただきました．ありがとうございます．

奈良先端科学技術大学院大学 情報科学研究科 ソフトウェア工学講座 メンバーの皆様には，研究や発表練習，それ以外の面でもさまざまなご支援を頂きました．学生生活を有意義なものにできたのも皆さんのおかげです．ありがとうございます．

最後に，これまでの人生を支えてくれた両親と妹に深く感謝いたします．

## 参考文献

- [1] Free Software Foundation, Inc. GNU General Public License Version 2, 1991. <http://www.gnu.org/copyleft/gpl.html>.
- [2] Tomohiro Ueno. The protest page to PocketMascot, September 2001. [http://members.jcom.home.ne.jp/tomohiro-ueno/About\\_PocketMascot/About\\_PocketMascot\\_e.html](http://members.jcom.home.ne.jp/tomohiro-ueno/About_PocketMascot/About_PocketMascot_e.html).
- [3] Leander Kahney. 『ペアー PC』の盗作？ マックエミュレーター 『チェリー OS』 (Wired News), October 2004. <http://hotwired.goo.ne.jp/news/technology/story/20041019301.html>.
- [4] Tatsuyoshi. LD-WBBR/B について (Tatsuyoshi tech diary), May 2004. <http://www.tatsuyoshi.net/toyota/tech/elecom/>.
- [5] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: A multi-linguistic token-based code clone detection system for large scale source code. *IEEE Trans. on Software Engineering*, Vol. 28, No. 7, pp. 654–670, 2002.
- [6] Haruaki Tamada, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto. Java birthmarks —detecting the software theft—. *IEICE Transactions on Information and Systems*, Vol. E88-D, No. 9, pp. 2148–2158, September 2005.
- [7] Business Software Alliance. Global software piracy study, June 2004. <http://www.bsa.org/globalstudy/>.
- [8] 有限会社パックスシステムズのソースコード盗用についてのページ, August 2002. <http://hp.vector.co.jp/authors/VA011396/>.
- [9] メディアカイト, 「ネットゲーせん」を延期, August 2002. <http://www.itmedia.co.jp/games/gsnews/0208/30/news08.html>.

- [10] 岡田大助. オープンソースソフトを家電店が“ 自社開発 ”と偽って販売 (cnet japan), February 2004. <http://japan.cnet.com/news/media/story/0,2000047715,20064430,00.htm>.
- [11] フリーソフトを“ 自社開発 ”と偽って販売, October 2003. <http://dayomon.fc2web.com/omaturi/open.html>.
- [12] オンラインソフト作家はパクリ屋だった・・・そんな話, October 2003. <http://wanderingdj.blogtribe.org/entry-9894b303326b08005c5fedc551611d54%.html>.
- [13] GPL違反指摘の「DVDコンバータ」、ソースコード開示 「CSS解除機能は搭載していない」(ITMedia), February 2003. [http://www.itmedia.co.jp/news/0302/13/njbt\\_03.html](http://www.itmedia.co.jp/news/0302/13/njbt_03.html).
- [14] PornView は GImageView のコードをパクリまくり (slashdot.jp), December 2002. <http://slashdot.jp/article.pl?sid=02/12/13/0713242>.
- [15] Sigma Designs、RMP4 に XVID からのコード流用を認める (PC Watch), August 2002. <http://www.watch.impress.co.jp/av/docs/20020828/sigma.htm>.
- [16] Epson Pulls Linux Software Following GPL Violations(slashdot.org), September 2002. <http://slashdot.org/article.pl?sid=02/09/11/2225212>.
- [17] エプソンコーワ、GPL 違反発覚も平素の対応に賛辞の声 (slashdot.jp), September 2002. <http://slashdot.jp/article.pl?sid=02/09/15/1535206>.
- [18] Ryan Singel. 再リリース『チェリー OS』、やはり『ペアー OS』を盗用?(Wired News), March 2005. <http://hotwired.goo.ne.jp/news/technology/story/20050311301.html>.

- [19] 米 Symantec、プログラムコード盗用を理由に米 McAfee を提訴, May 1997. <http://pc.watch.impress.co.jp/docs/article/970501/steal.htm>.
- [20] 土木設計ソフトのコピー販売で損害賠償を求めて提訴 (ITMedia), October 2000. <http://www.itmedia.co.jp/news/0010/02/accs.html>.
- [21] Eric Raymond and Rob Landley. OSI position paper on the SCO-vs.-IBM complaint, May 2004. <http://www.opensource.org/sco-vs-ibm.html>.
- [22] Akito Monden. jmark: A lightweight tool for watermarking java class files, 2002. <http://se.aist-nara.ac.jp/jmark/>.
- [23] Christian Collberg. Sandmark: A tool for the study of software protection algorithms, 2000. <http://www.cs.arizona.edu/sandmark/>.
- [24] Christian Collberg and Clark Thomborson. Software watermarking: Models and dynamic embeddings. In *Proc. Principles of Programming Languages 1999, POPL'99*, pp. 311–324, January 1999. San Antonio, TX.
- [25] Christian Collberg and Clark Thomborson. Watermarking, tamper-proofing, and obfuscation - tools for software protection. *IEEE Transactions on Software Engineering*, Vol. 28, No. 8, pp. 735–746, August 2002.
- [26] Clark Thomborson, Jasvir Nagra, Ram Somaraju, and Charles He. Tamper-proofing software watermarks. In *Proc. 2nd workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation*, Vol. 32, pp. 27–36, Australian Computer Society, Inc., 2004. Dunedin, New Zealand.
- [27] Alex Aiken. MOSS: A system for detecting software plagiarism, June 2004. <http://www.cs.berkeley.edu/~aiken/moss.html>.
- [28] Xin Chen, Brent Francia, Ming Li, Brian Mckinnon, and Amit Seker. SID plagiarism detection, Dec 2003. <http://genome.math.uwaterloo.ca/SID/>.

- [29] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*, Vol. 8, No. 11, pp. 1016–1038, November 2002.
- [30] Michael J. Wise. YAP3: Improved detection of similarities in computer program and other texts. In *Proc. 27 SIGCSE Technical Symposium on Computer Science Education*, pp. 130–134, 1996. Philadelphia, Pennsylvania, United States.
- [31] Ira D. Baxter, Andrew Yahin, Leonardo M. De Moura, Marcelo Sant’Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *ICSM: the International Conference on Software Maintenance*, pp. 368–377, 1998.
- [32] Derrick Grover, editor. *The protection of computer software —its technology and applications Second edition*. The British Computer Society Monographs in Informatics Cambridge University Press, May 1992.
- [33] Ivan Krsul and Eugene H. Spafford. Authorship analysis: Identifying the author of a program. *Computers and Security*, Vol. 16, No. 3, pp. 233–257, 1997.
- [34] Eugene H. Spafford and Stephen A. Weeber. Software forensics: Can we track code to its authors? *Computers and Security*, Vol. 12, No. 6, pp. 585–595, 1993.
- [35] 玉田春昭, 神崎雄一郎, 中村匡秀, 門田暁人, 松本健一. Java クラスファイルからプログラム指紋を抽出する方法の提案. 電子情報通信学会 技術研究報告 情報セキュリティ研究会, 第 ISEC 2003-29 巻, pp. 127–133, July 2003.
- [36] Haruaki Tamada, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto. Detecting the theft of programs using birthmarks. Information Science Technical Report NAIST-IS-TR2003014 ISSN 0919-9527, Graduate School of Information Science, Nara Institute of Science and Technology, November 2003.

- [37] Haruaki Tamada, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto. Design and evaluation of birthmarks for detecting theft of java programs. In *Proc. IASTED International Conference on Software Engineering (IASTED SE 2004)*, pp. 569–575, February 2004. Innsbruck, Austria.
- [38] 玉田春昭. バースマークと動的名前解決によるソフトウェアプロテクション. 博士論文, 奈良先端科学技術大学院大学 情報科学研究科, March 2006.
- [39] 福島和英, 田端利宏, 櫻井幸一. Java バイトコードの静的解析によるプログラム指紋の抽出方法. 情報処理学会研究報告, コンピュータセキュリティ研究会 2003-126, 第 126 巻, pp. 81–86, December 2003.
- [40] 福島和英, 田端利宏, 櫻井幸一. 動的解析を用いた java クラスファイルのプログラム指紋抽出法. 暗号と情報セキュリティシンポジウム 2004 (SCIS 2004), 第 1 巻, pp. 1327–1332, January 2004.
- [41] Ginger Myles and Christian Collberg. K-gram based software birthmarks. In *Proc. the 2005 ACM Symposium on Applied Computing*, pp. 314–318, March 2005. Santa Fe, New Mexico.
- [42] Ginger Myles and Christian Collberg. Detecting software theft via whole program path birthmarks. In *Proc. Information Security 7th International Conference, ISC 2004*, Vol. 3225, pp. 404–415. Springer-Verlag GmbH, September 2004. Palo Alto, CA, USA.
- [43] CodingArt Pty Ltd. Codeshield java byte code obfuscator, 1999. <http://www.codingart.com/codeshield.html>.
- [44] 門田暁人, 高田義広, 鳥居宏次. ループを含むプログラムを難読化する方法の提案. 電子情報通信学会論文誌 D-I, Vol. J80-D-I, No. 7, pp. 644–652, July 1997.
- [45] Microsoft Corporation. Microsoft Windows 32 ビット API リファレンス, 1993.



- [46] Jeffrey Richter, 長尾高弘 ( 訳 ). Advanced Windows 改訂第 4 版. Microsoft Press, ASCII, 2001.
- [47] 岡本圭司. K2BTK K2 Birthmark Tool kit, March 2005. <http://se.naist.jp/K2BTK/>.
- [48] Microsoft Corporation. Microsoft Platform SDK. <http://www.microsoft.com/msdownload/platformsdk/sdkupdate/>.
- [49] haseta. Super Tag Editor 改, February 2004. <http://hp.vector.co.jp/authors/VA012911/mp3DB/ste.html>.
- [50] Mercury. Super Tag Editor, December 2001. <http://www5.wisnet.ne.jp/~mercury/supertag/index.html>.
- [51] P's soft. MPEG ID3 TAG Editor つゆたく, March 2004. <http://www.lares.dti.ne.jp/~mk3/Akmssoft/tuyutag.htm>.
- [52] 黒田徹. Teatime, July 2002. <http://hp.vector.co.jp/authors/VA011396/>.
- [53] Florian Heidenreich. Mp3tag - the universal Tag Editor and more, November 2004. <http://www.mp3tag.de/en/index.html>.
- [54] 中川哲裕. Wdiff, May 1998. <http://www.vector.co.jp/soft/win95/util/se057654.html>.
- [55] Microsoft. Word2000, 1999. <http://www.microsoft.com/japan/office/previous/2000/word/default.msp>.
- [56] JustSystems. 一太郎 13, 2003. <http://www.justsystem.co.jp/software/dt/taro13/index.html>.
- [57] JustSystems. 一太郎 2004, 2004. <http://www.ichitaro.com/2004/>.

- [58] 森山修, 古江岳大, 遠山毅, 松本勉. API 関数呼出履歴によるソフトウェア動的バースマークの一方式. 信学技報, 第 106 巻, pp. 77–84, September 2006.
- [59] Tom White. Jazzy に勝るものはありません, September 2004. [http://www-06.ibm.com/jp/developerworks/java/041217/j\\_j-jazzy.html](http://www-06.ibm.com/jp/developerworks/java/041217/j_j-jazzy.html).

## 付録

### A. 実験1の各ソフトウェアのAPI呼び出し回数

5.1の実験1における，各ソフトウェアのAPI呼び出し回数は表22の通り．

表 22 各ソフトウェアのAPI呼び出し回数

ソフトウェア名	API呼び出し回数
STE	13,286
STE-Ext	13,447
Tuyutag	10,834
TeaTime	1,647
Mp3tag	191,783

### B. 7.5の実験的評価における実行系列バースマークの長さ

7.5の実験的評価における，それぞれの実行系列バースマークの長さ（API呼び出し回数）は以下の通り．(2)の実行系列バースマークは，ファイルを(1)と比較してAPI呼び出しが160,029回（14.8%）増加している．この大部分は「適切なファイルを開く」操作により増加したものであると考えられる．

表 23 実験的評価における実行系列バースマークの長さ

操作	API呼び出し回数
(1) 起動 終了	1,082,409
(2) 起動 適切なファイルを開く 終了	1,242,438